

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

IPA TECHNOLOGIES, INC.,)	
)	PUBLIC VERSION
)	
Plaintiff,)	The Honorable
)	Richard G. Andrews
v.)	
)	
AMAZON.COM, INC., and AMAZON DIGITAL)	
SERVICES, LLC,)	Civil Action 1:16-CV-01266-RGA
)	
Defendants.)	JURY TRIAL DEMANDED
)	

**DECLARATION OF VIGEN SALMASTLIAN IN SUPPORT OF ANSWERING BRIEF
IN OPPOSITION TO IPA TECHNOLOGIES, INC.'S MOTION FOR SUMMARY
JUDGMENT OF NO INVALIDITY BASED ON THE OAA SYSTEM**

(VOLUME I OF II)

Dated: July 30, 2021

Of Counsel:

J. David Hadden
Saina S. Shamilov
Ravi Ranganath
Vigen Salmastlian
FENWICK & WEST LLP
801 California Street Mountain
View, CA 94041 (650)
988-8500

Todd R. Gregorian
Sapna S. Mehta
Eric B. Young
Min Wu
FENWICK & WEST LLP
555 California Street, 12th Floor
San Francisco, CA 94104 (415)
875-2300

ASHBY & GEDDES

Steven J. Balick (#2114)
Andrew C. Mayo (#5207)
500 Delaware Avenue, 8th Floor
P.O. Box 1150
Wilmington, DE 19899
(302) 654-1888
sbalick@ashbygeddes.com
amayo@ashbygeddes.com

*Attorneys for Defendants Amazon.com, Inc.
and Amazon Digital Services, LLC*

I, Vigen Salmastlian, declare as follows:

1. I am a licensed attorney admitted in the State of California and admitted *pro hac vice* to this Court. I am an associate with the law firm of Fenwick & West LLP, counsel of record for defendants Amazon.com, Inc. and Amazon Digital Services LLC (collectively, “Amazon”). I have personal knowledge of the facts in this declaration and can competently testify to those facts.

2. Attached hereto as **Exhibit 1** is a true and correct copy of U.S. Patent No. 6,851,115.

3. Attached hereto as **Exhibit 2** is a true and correct copy of U.S. Patent No. 7,096,560.

4. Attached hereto as **Exhibit 3** is a true and correct copy of the Errata to Opening Expert Report of Katia P. Sycara, Ph.D. dated March 12, 2021.

5. Attached hereto as **Exhibit 4** is a true and correct copy of the declaration of David L. Martin filed in this case July 6, 2021.

6. Attached hereto as **Exhibit 5** is a true and correct copy of the declaration of Raymond C. Perrault filed in this case on July 6, 2021.

7. Attached hereto as **Exhibit 6** is a true and correct copy of the deposition transcript of Adam Cheyer in this case dated November 9, 2020.

8. Attached hereto as **Exhibit 7** is a true and correct copy of the deposition transcript of David Martin in this case dated November 11, 2020.

9. Attached hereto as **Exhibit 8** is a true and correct copy of the deposition transcript of Douglas Moran in this case dated November 16, 2020.

10. Attached hereto as **Exhibit 9** is a true and correct copy of the deposition transcript of Luc Julia in this case dated December 1, 2020.

11. Attached hereto as **Exhibits 10** is a true and correct copy of the declaration of Luc Julia produced in this case on December 3, 2020.

12. Attached hereto as **Exhibits 11** is a true and correct copy of the document subpoena served on SRI International in this case on July 18, 2019.

13. Attached hereto as **Exhibit 12** is a true and correct copy of the curriculum vitae of Douglas Moran, which is Exhibit 1 from the deposition of Douglas Moran in this case.

14. Attached hereto as **Exhibit 13** is a true and correct copy of the declaration of Douglas B. Moran from IPR2019-00728. *Google LLC v. IPA Techs. Inc.*, IPR2019-00728, Paper 1, Ex. 1007 (PTAB Feb. 26, 2019).

15. Attached hereto as **Exhibit 14** is a true and correct copy of the file [REDACTED]

[REDACTED]

[REDACTED]

16. Attached hereto as **Exhibit 15** is a true and correct copy of a May 16, 1997 internet archive capture of the webpage <http://www.ai.sri.com/~oaa/distrib.html>, which is Exhibit 26 from the deposition of Adam Cheyer in this case.

17. Attached hereto as **Exhibit 16** is a true and correct copy of the “OAA-USERS Mailing List Archive by date” from the webpage <http://www.ai.sri.com/~oaa/distribution/MailArchive/oaa-users/date.html>, which is Exhibit 31 from the deposition of Adam Cheyer in this case.

18. Attached hereto as **Exhibit 17** is a true and correct copy of Exhibit 6 to the declaration of Raymond C. Perrault filed July 6, 2021.

19. Attached hereto as **Exhibit 18** is a true and correct copy of the Reply Expert Report of Katia P. Sycara, Ph.D. dated May 7, 2021.

20. Attached hereto as **Exhibit 19** is a true and correct copy of the webpage <http://www.adam.cheyer.com/demo-office.html>, which is Exhibit 9 from the deposition of Adam Cheyer in this case.

21. Attached hereto as **Exhibit 20** is a true and correct copy of a video produced as DEFS_IPA_PA00021314 titled “Open Agent Architecture Automated Office Demonstration.”

22. Attached hereto as **Exhibit 21** is a true and correct copy of

[REDACTED]

[REDACTED]

23. Attached hereto as **Exhibit 22** is a true and correct copy of

[REDACTED]

[REDACTED]

24. Attached hereto as **Exhibit 23** is a true and correct copy of [REDACTED]

[REDACTED]

25. Attached hereto as **Exhibit 24** is a true and correct copy of a publication produced as DEFS_IPA_PA00034264-279, titled “MVIEW: Multimodal Tools for the Video Analyst,” which is Exhibit 4 from the deposition of Adam Cheyer.

26. Attached hereto as **Exhibit 25** is a true and correct copy of a presentation produced as DEFS_IPA_PA00021059-1136, titled “PAAM ’98 Tutorial Building and Using Practical Agent Applications.”

27. Attached hereto as **Exhibit 26** is a true and correct copy of the deposition transcript of Adam Cheyer from IPR2019-00728. *Google LLC v. IPA Technologies Inc.*, IPR2019-00728, Paper 2037 (PTAB December 17, 2019).

28. Attached hereto as **Exhibit 27** is a true and correct copy of the deposition transcript of David Martin from IPR2019-00728. *Google LLC v. IPA Technologies Inc.*, IPR2019-00728, Paper 2038 (PTAB December 18, 2019).

29. Attached hereto as **Exhibit 28** is a true and correct copy of an excerpt of the certified prosecution history of U.S. Patent No. 6,851,115.

30. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

31.

[REDACTED]

[REDACTED]

[REDACTED]

32. Reproduced below is a true and correct excerpt of the

[REDACTED]

[REDACTED]

33. Reproduced below are true and correct

[REDACTED]

[REDACTED]

[REDACTED]

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct. Executed in Saratoga, California on July 30, 2021.

/s/ Vigen Salmastlian

Vigen Salmastlian

Exhibit 1

U 8037680

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

March 11, 2020


**THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM
THE RECORDS OF THIS OFFICE OF:**

U.S. PATENT: 6,851,115

ISSUE DATE: February 01, 2005

**By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office**




R GLOVER
Certifying Officer



US006851115B1

(12) **United States Patent**
Cheyer et al.

(10) **Patent No.: US 6,851,115 B1**
(45) **Date of Patent: Feb. 1, 2005**

- (54) **SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS**
- (75) Inventors: **Adam J. Cheyer**, Palo Alto, CA (US);
David L. Martin, Santa Clara, CA (US)
- (73) Assignee: **SRI International**, Menlo Park, CA (US)
- (*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.
- (21) Appl. No.: **09/225,198**
- (22) Filed: **Jan. 5, 1999**
- (51) Int. Cl.⁷ **G06F 9/54**
- (52) U.S. Cl. **719/317; 709/202; 717/114**
- (58) Field of Search **719/317; 709/202, 709/207, 317; 717/114**

(56) **References Cited**

U.S. PATENT DOCUMENTS

- 5,638,494 A * 6/1997 Pinard et al. 709/202
- 5,802,396 A * 9/1998 Gray 710/20
- 5,960,404 A * 9/1999 Chaar et al. 705/11
- 6,047,053 A 4/2000 Miner et al.
- 6,212,649 B1 * 4/2001 Yalowitz et al. 714/31
- 6,216,173 B1 * 4/2001 Jones et al. 135/77
- 6,256,771 B1 7/2001 O'Neil et al.
- 6,338,081 B1 * 1/2002 Furusawa et al. 709/202
- 6,411,684 B1 6/2002 Cohn et al.
- 6,484,155 B1 * 11/2002 Kiss et al. 706/46
- 2001/0039562 A1 * 11/2001 Sato 709/202
- 2003/0167247 A1 * 9/2003 Masuoka 706/46

OTHER PUBLICATIONS

- Cheyer, Adam. "Mechanisms of Cooperation." Oct. 19, 1998.*
- DeVoe, Deborah. "SRI distributed agents promise flexibility." InfoWorld. Dec. 30, 1996.*

- Sycara, Katia et al. "Distributed Intelligent Agents." IEEE. Dec. 1996.*
- Nwana, Hyacinth et al. "Software Agent Technologies". BT Technology Journal. 1996.*
- Busetta, Paolo et al. "The BDIM Agent Toolkit Design." 1997.*
- Mayfield, James et al. "Desiderata for Agent Communication Languages." Mar. 27-29, 1995.*
- Khedro, Taha et al. Concurrent Engineering through Interoperable Software Agents. Aug. 1994.*
- Moran et al. "Multimodal User Interfaces in the Open Agent Architecture." Proceedings of the International Conference on Intelligent User Interfaces. 6-9/1997.*
- Martin, David et al. "The Open Agent Architecture: A Framework for Building Distributed Software Systems." Oct. 19, 1998.*
- Wilkins, David et al. "Multiagent Planning Architecture." SRI International. Dec. 8, 1997.*
- Moran, Douglas B. and Cheyer, Adam J., "Intelligent Agent-based User Interfaces", Article Intelligence center, SRI International.
- Martin, David L., Cheyer, Adam J. and Moran, Douglas B., "Building Distributed Software Systems with the Open Agent Architecture".

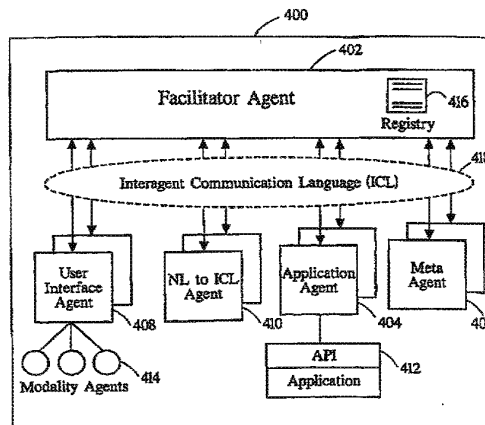
(List continued on next page.)

Primary Examiner—Lewis A. Bullock, Jr.
(74) *Attorney, Agent, or Firm*—Perkins Coie LLP

(57) **ABSTRACT**

A highly flexible, software-based architecture is disclosed for constructing distributed systems. The architecture supports cooperative task completion by flexible and autonomous electronic agents. One or more facilitators are used to broker communication and cooperation among the agents. The architecture provides for the construction of arbitrarily complex goals by users and service-requesting agents. Additional features include agent-based provision of multi modal interfaces, including natural language.

89 Claims, 16 Drawing Sheets



US 6,851,115 B1

Page 2

OTHER PUBLICATIONS

Cohen, Philip R. and Cheyer, Adam, SRI International, Wang, Michelle, Stanford University, BAEG, Soon Cheol, ETRI, "An Open Agent Architecture".

Julia, Luc E. and Cheyer, Adam J., SRI International "Cooperative Agents and Recognition Systems (CARS) for Drivers and Passengers".

Moran, Douglas B., Cheyer, Adam J., Julia, Luc E., Martin, David L., SRI International, and Park, Sangkyu, Electronics and Telecommunications Research Institute, "Multimodal User Interfaces in the Open Agent Architecture".

Cheyer, Adam and Lulia, Luc, SRI International "Multimodal Maps: An Agent-based Approach".

Cutkosky, Mark R., Engelmores, Robert S., Fikes, Richard E., Genesereth, Michael R., Gruber, Thomas R., Stanford University, Mark, William, Lockheed Palo Alto Research Labs, Tenenbaum, Jay M., Weber, Jay C., Enterprise Integration Technologies, "An Experiment in Integrating Concurrent Engineering Systems".

Martin, David L., Cheyer, Adam, SRI International, LEE, Gowang-Lo, ETRI, "Development Tools for the Open Agent Architecture", The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96), London, Apr. 1996.

Cheyer, Adam, Martin, David and Moran, Douglas, "The Open Agent architecture™", SRI International, AI Center.

Dejima, Inc., <http://www.dejima.com/>.

Cohen, Philip R, Cheyer, Adam, Wang, Michelle, Stanford University, BAEG, Soon Cheol ETRI: "An Open Agent Architecture," AAAI Spring Symposium, pp. 1-8, Mar. 1994.

Martin, David; Oohama, Hiroki; Moran, Douglas; Cheyer, Adam; "Information Brokering in an Agent Architecture," Proceeding of the 2nd International Conference on Practical Application of Intelligent Agents & Multi-Agent Technology, London, Apr. 1997.

* cited by examiner

U.S. Patent

Feb. 1, 2005

Sheet 1 of 16

US 6,851,115 B1

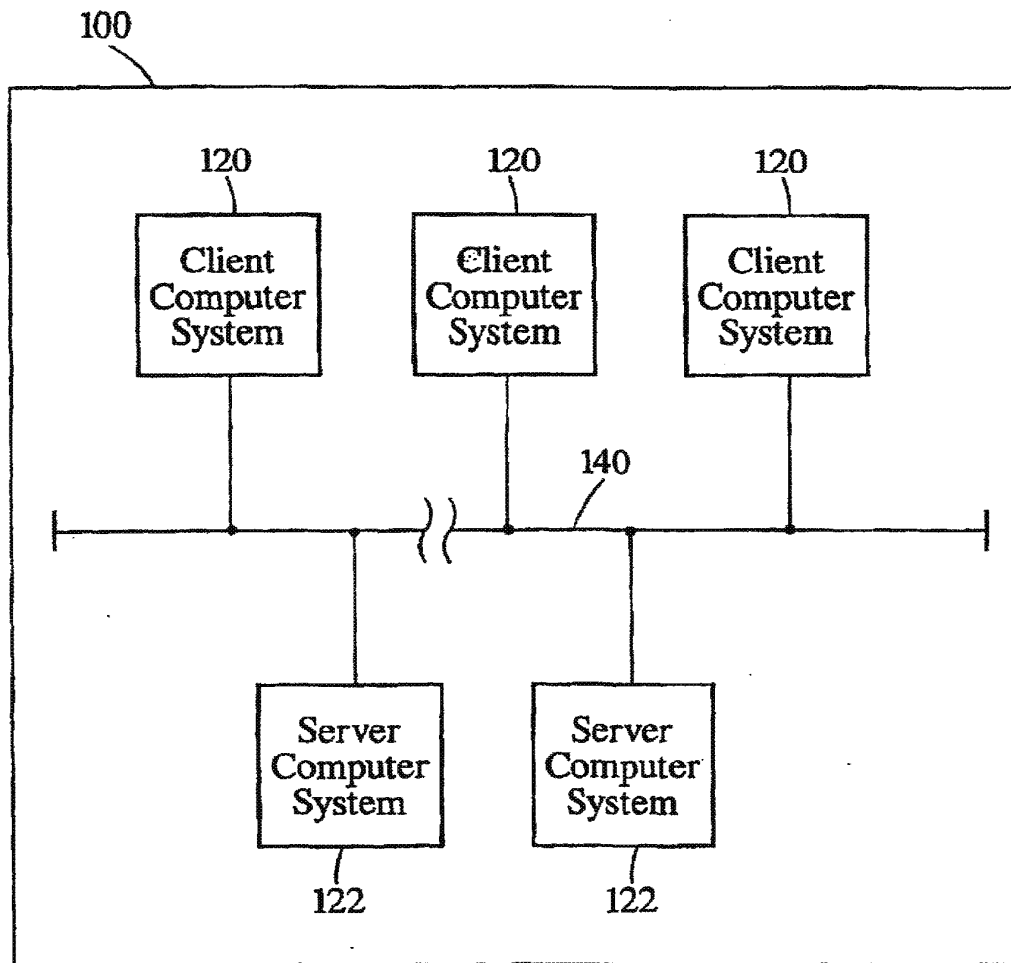


Fig. 1
(Prior Art)

U.S. Patent

Feb. 1, 2005

Sheet 2 of 16

US 6,851,115 B1

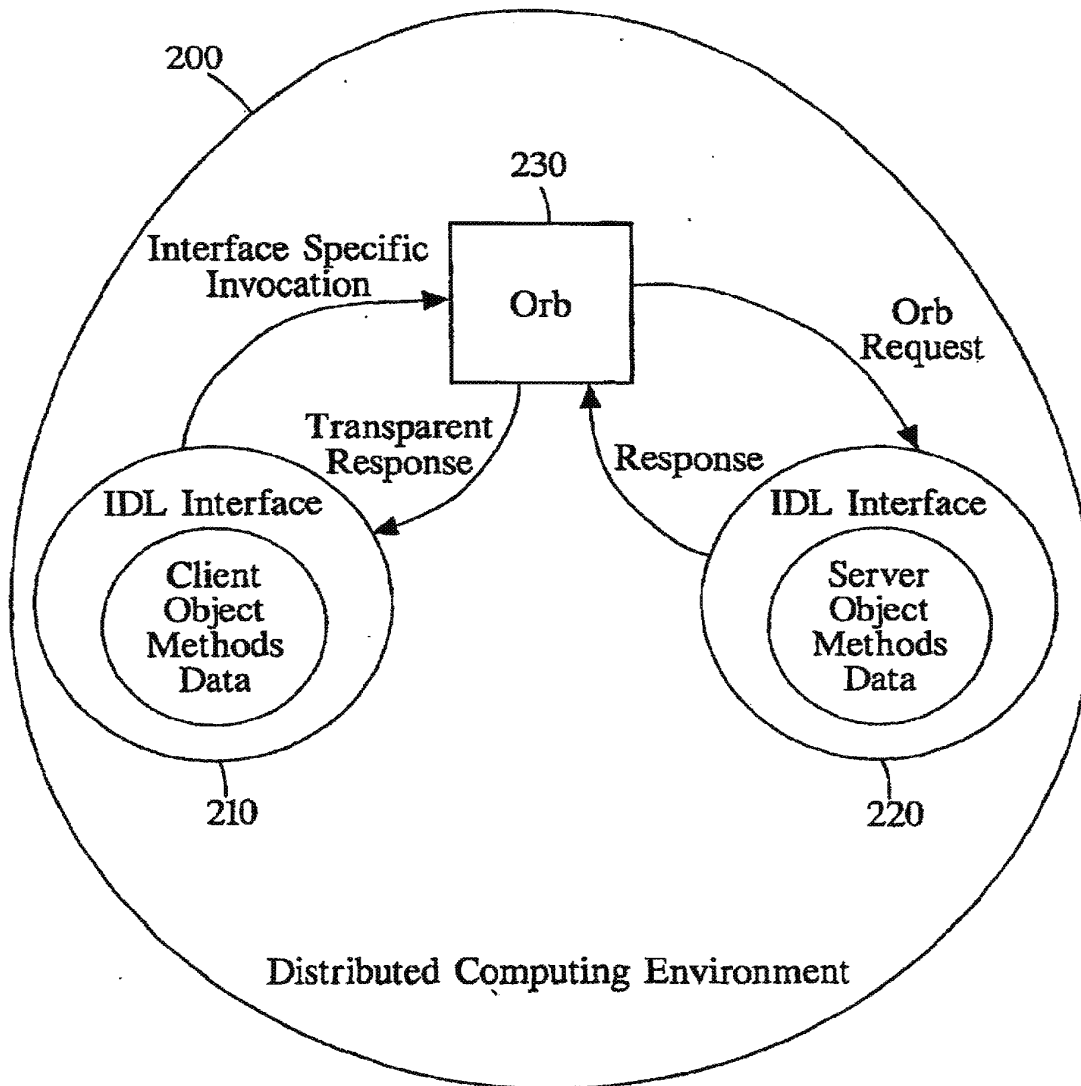


Fig. 2
(Prior Art)

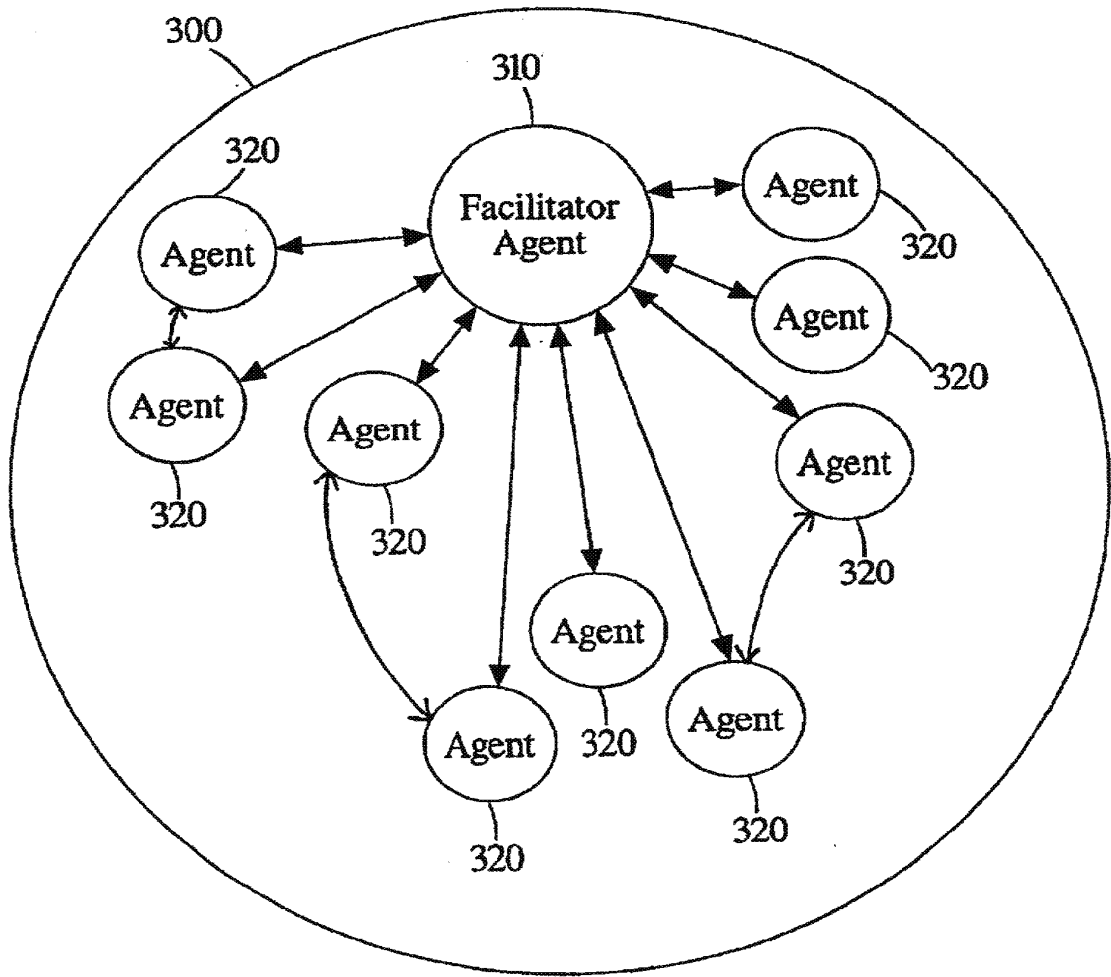


Fig. 3

U.S. Patent

Feb. 1, 2005

Sheet 4 of 16

US 6,851,115 B1

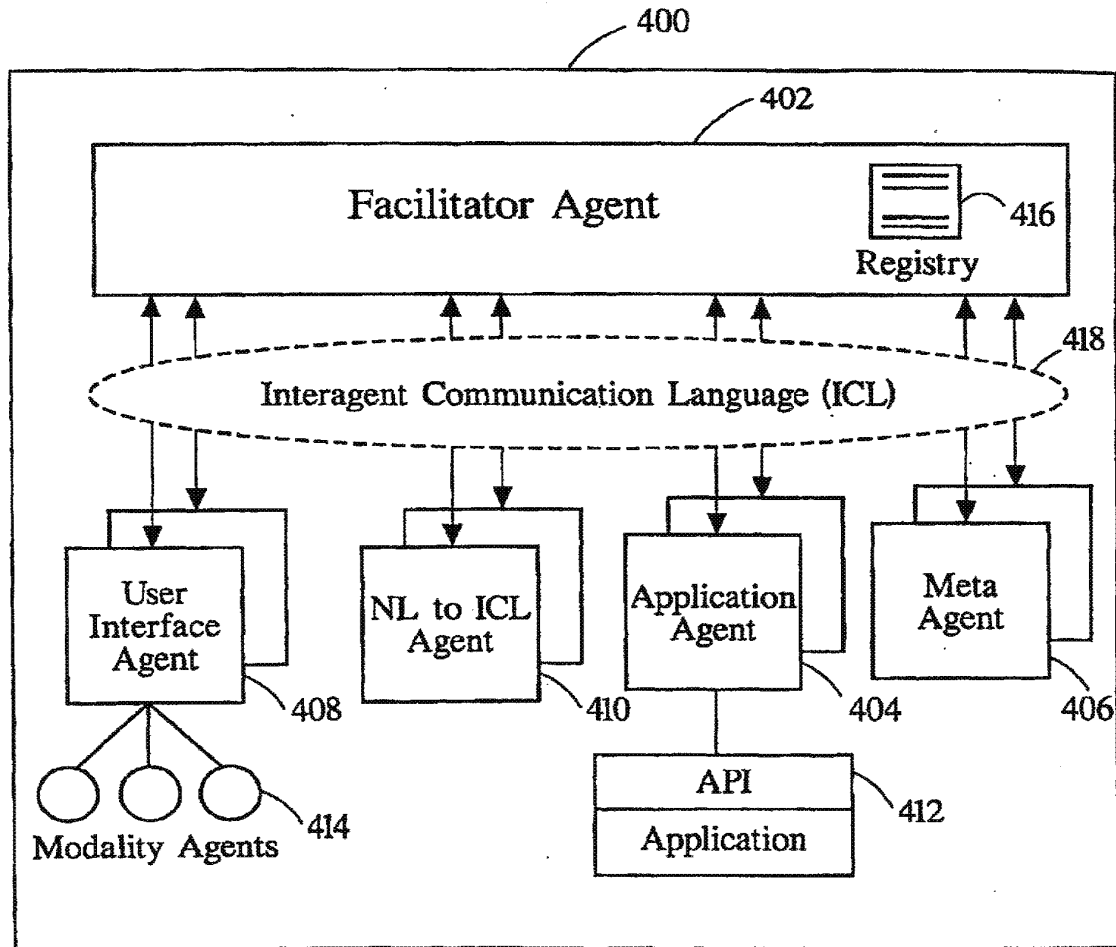


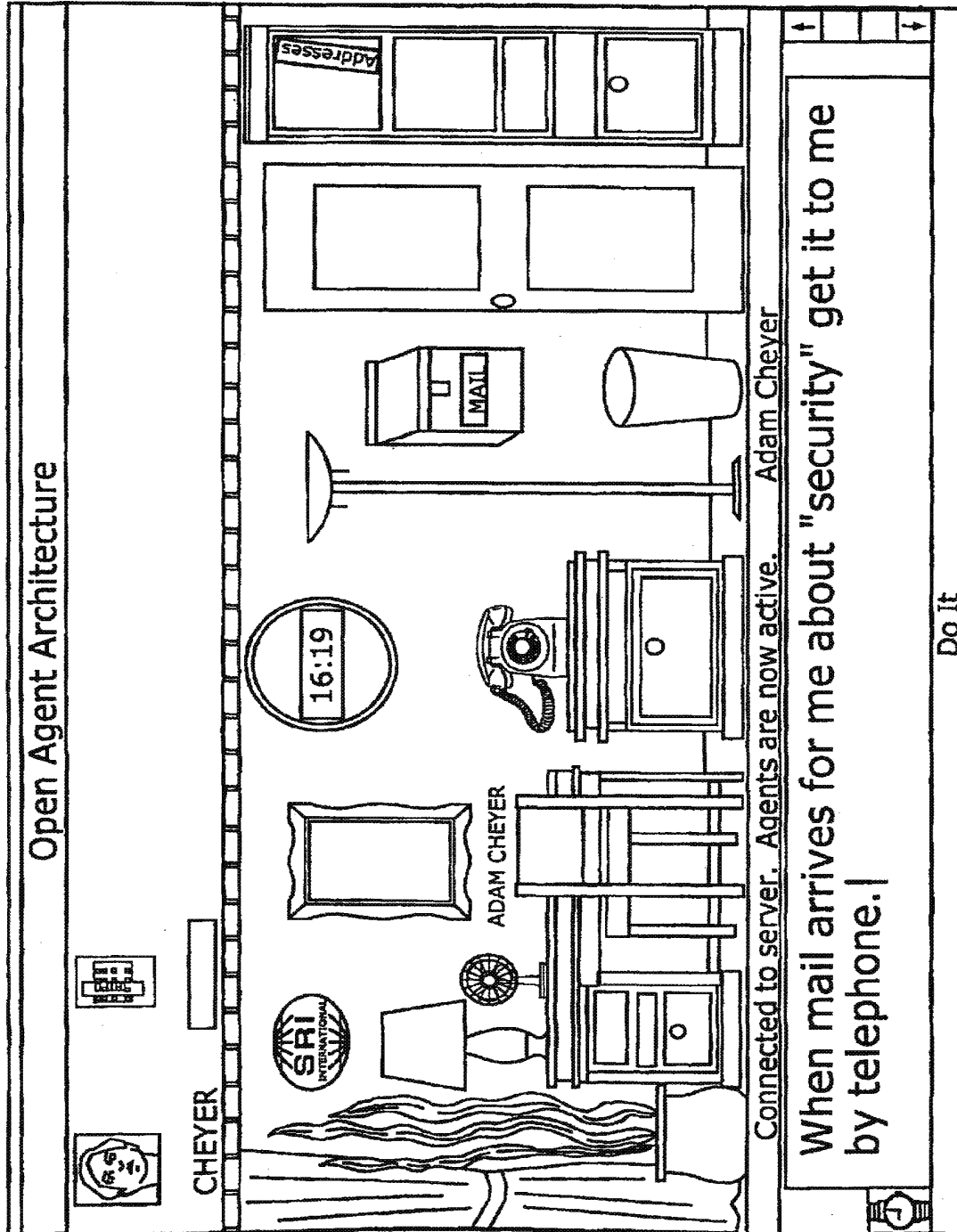
Fig. 4

U.S. Patent

Feb. 1, 2005

Sheet 5 of 16

US 6,851,115 B1



U.S. Patent

Feb. 1, 2005

Sheet 6 of 16

US 6,851,115 B1

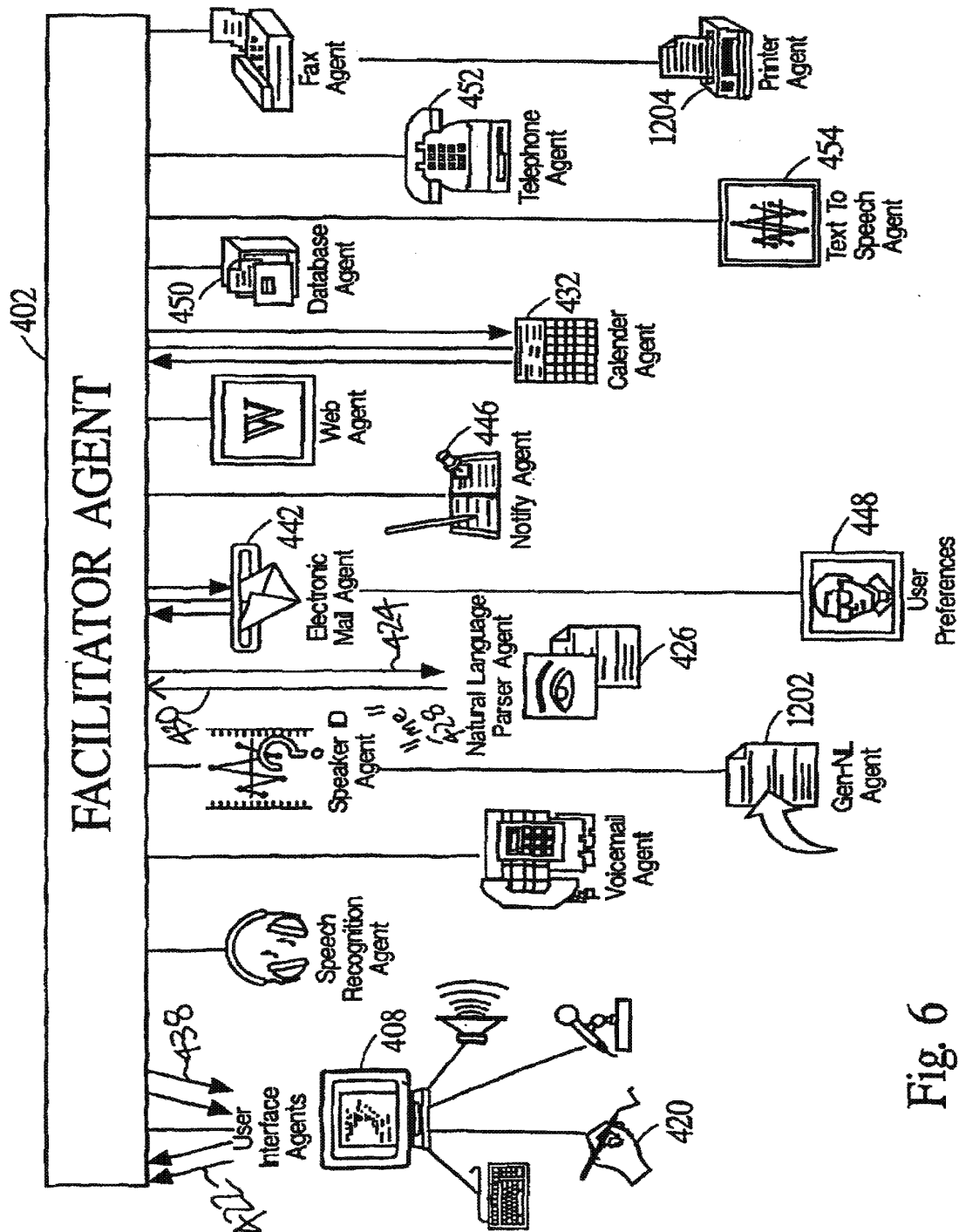


Fig. 6

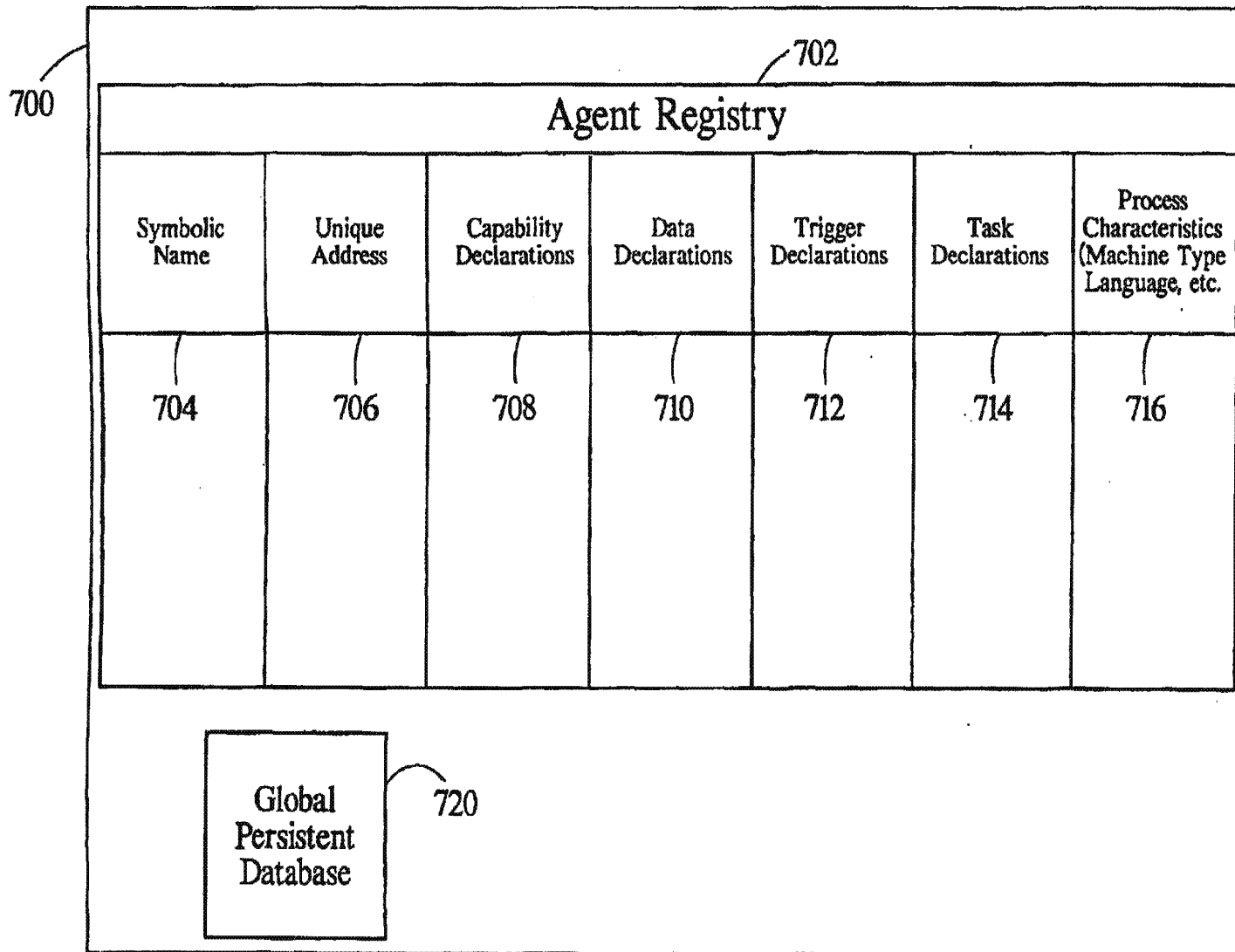


Fig. 7

U.S. Patent

Feb. 1, 2005

Sheet 8 of 16

US 6,851,115 B1

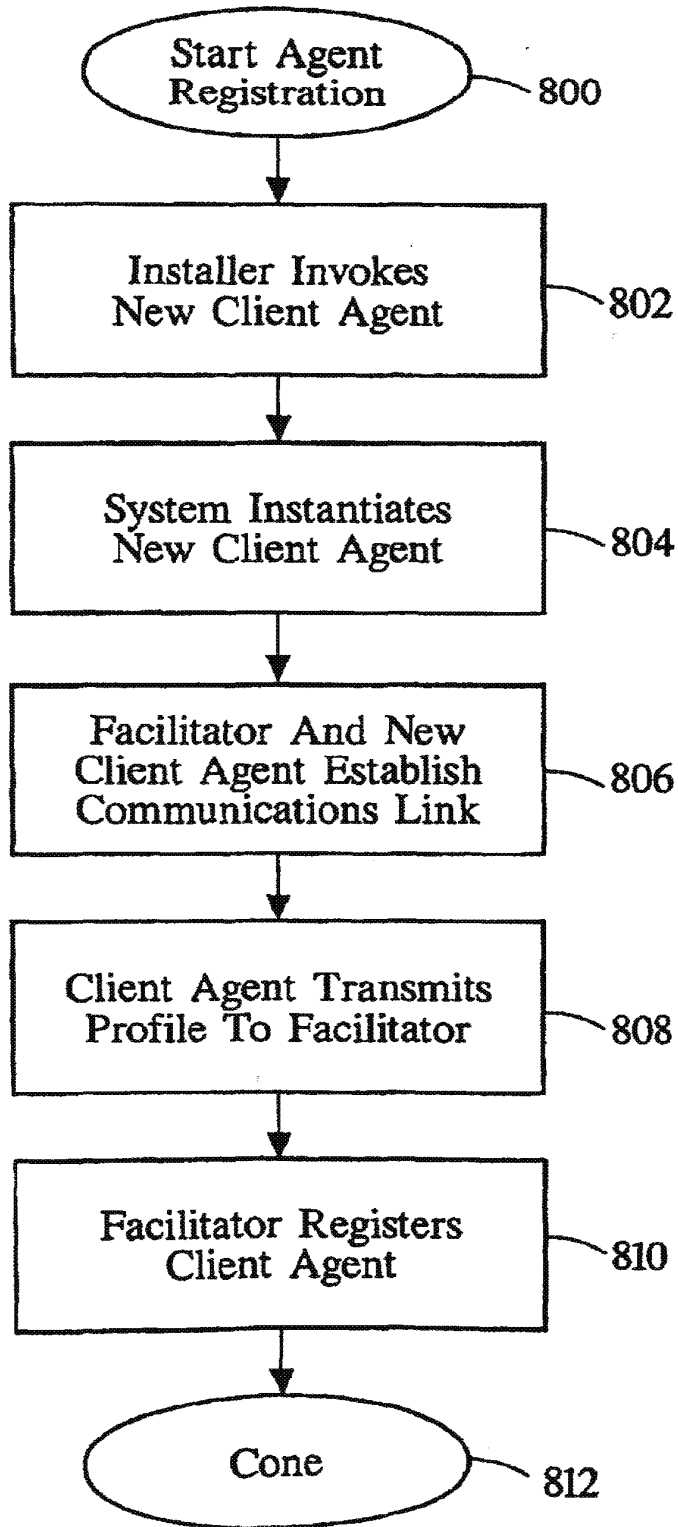


Fig. 8

U.S. Patent

Feb. 1, 2005

Sheet 9 of 16

US 6,851,115 B1

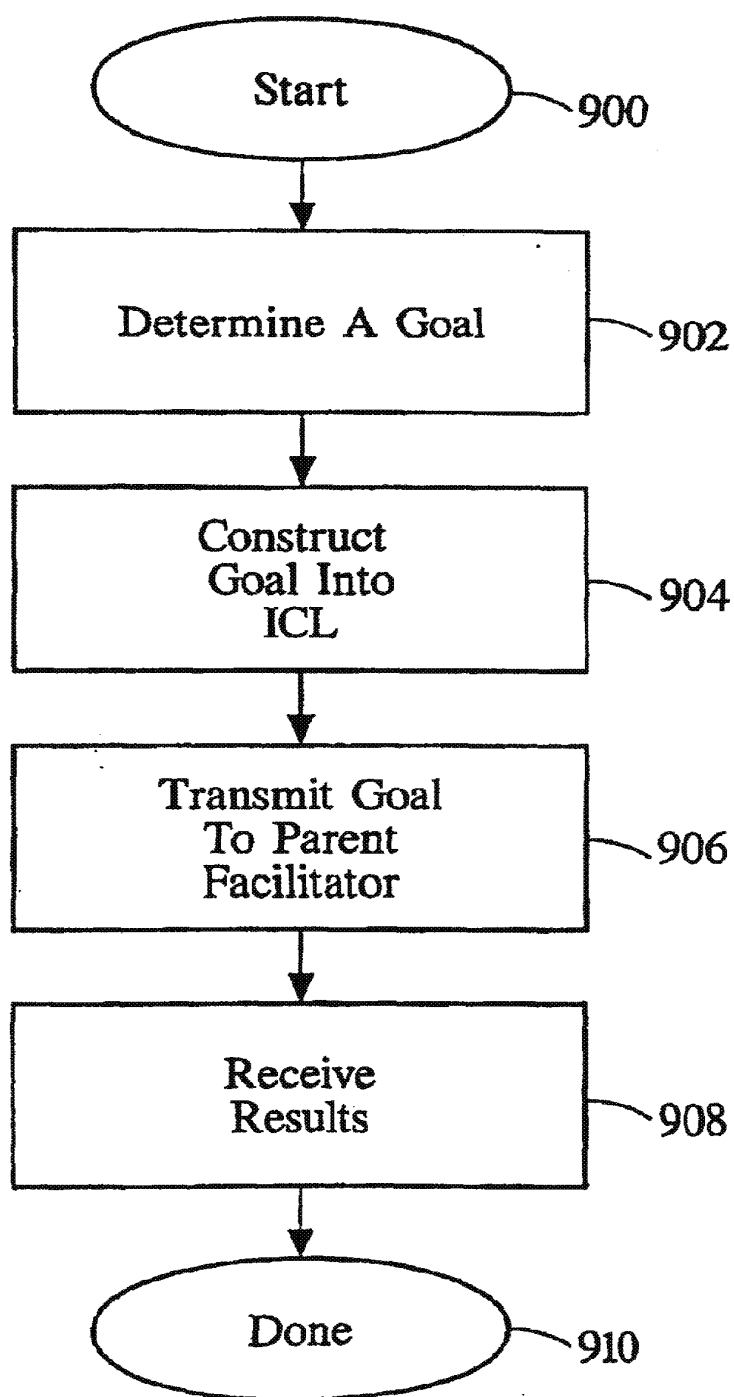


Fig. 9

U.S. Patent

Feb. 1, 2005

Sheet 10 of 16

US 6,851,115 B1

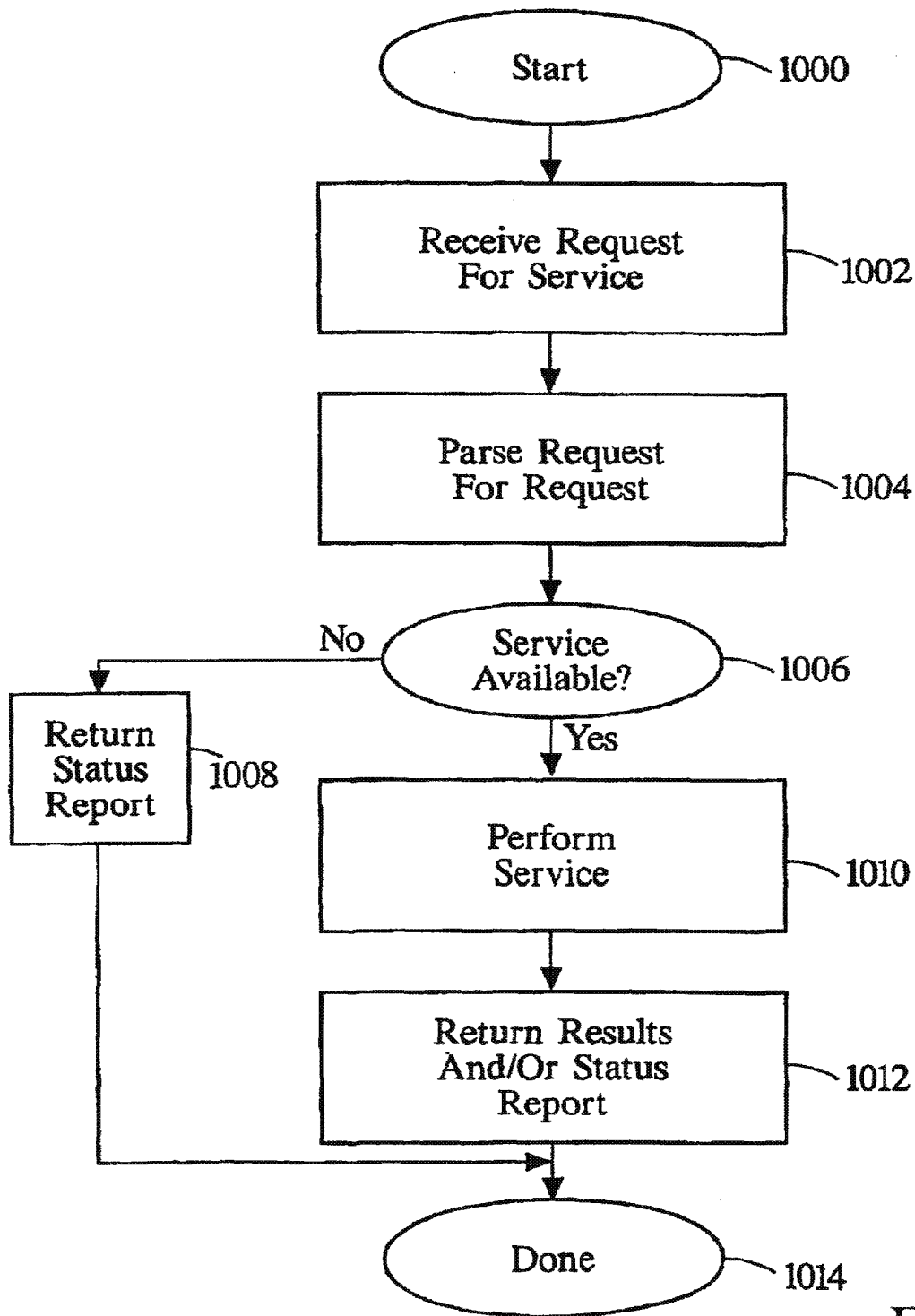


Fig. 10

U.S. Patent

Feb. 1, 2005

Sheet 11 of 16

US 6,851,115 B1

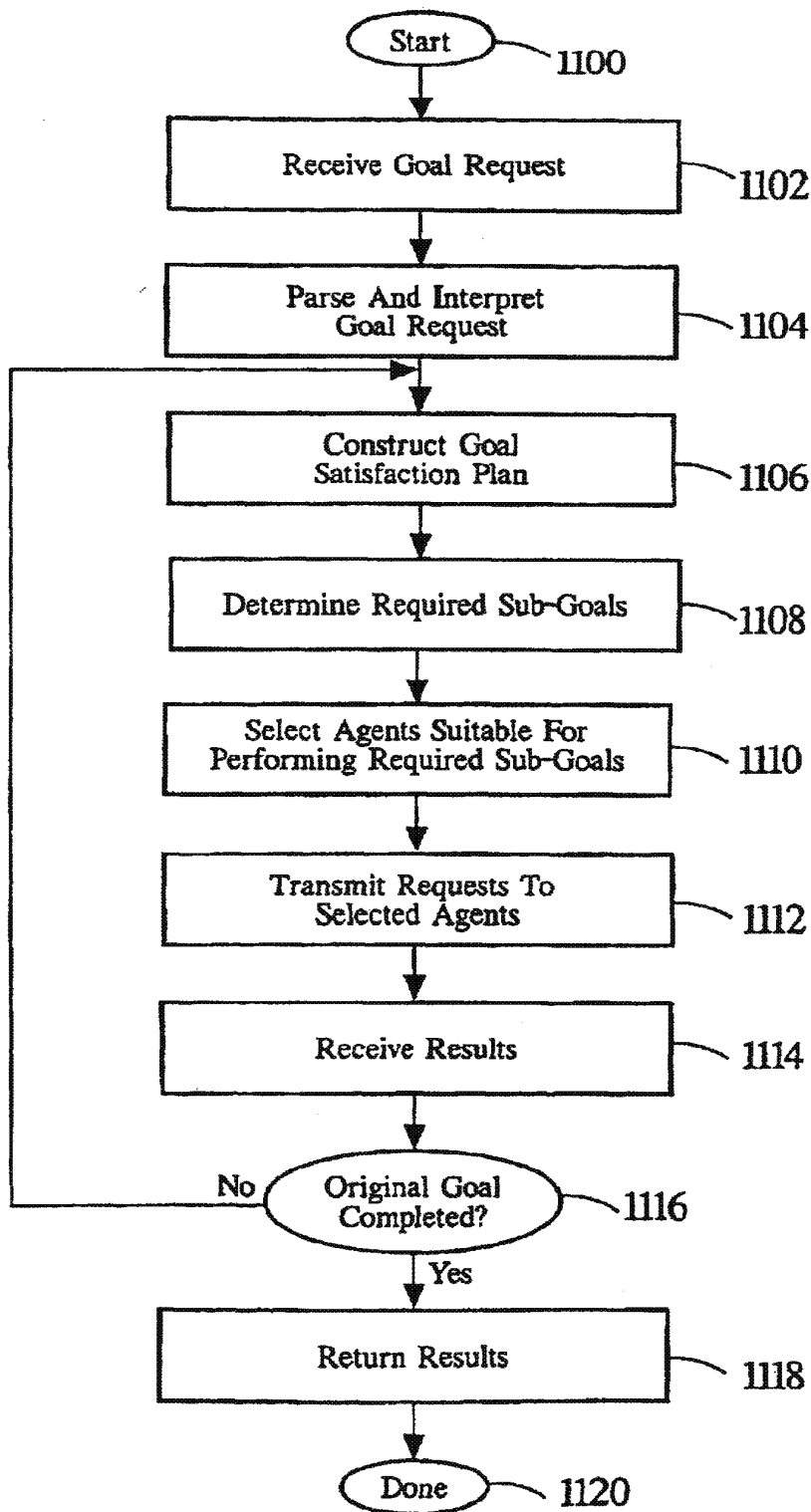


Fig. 11

U.S. Patent

Feb. 1, 2005

Sheet 12 of 16

US 6,851,115 B1

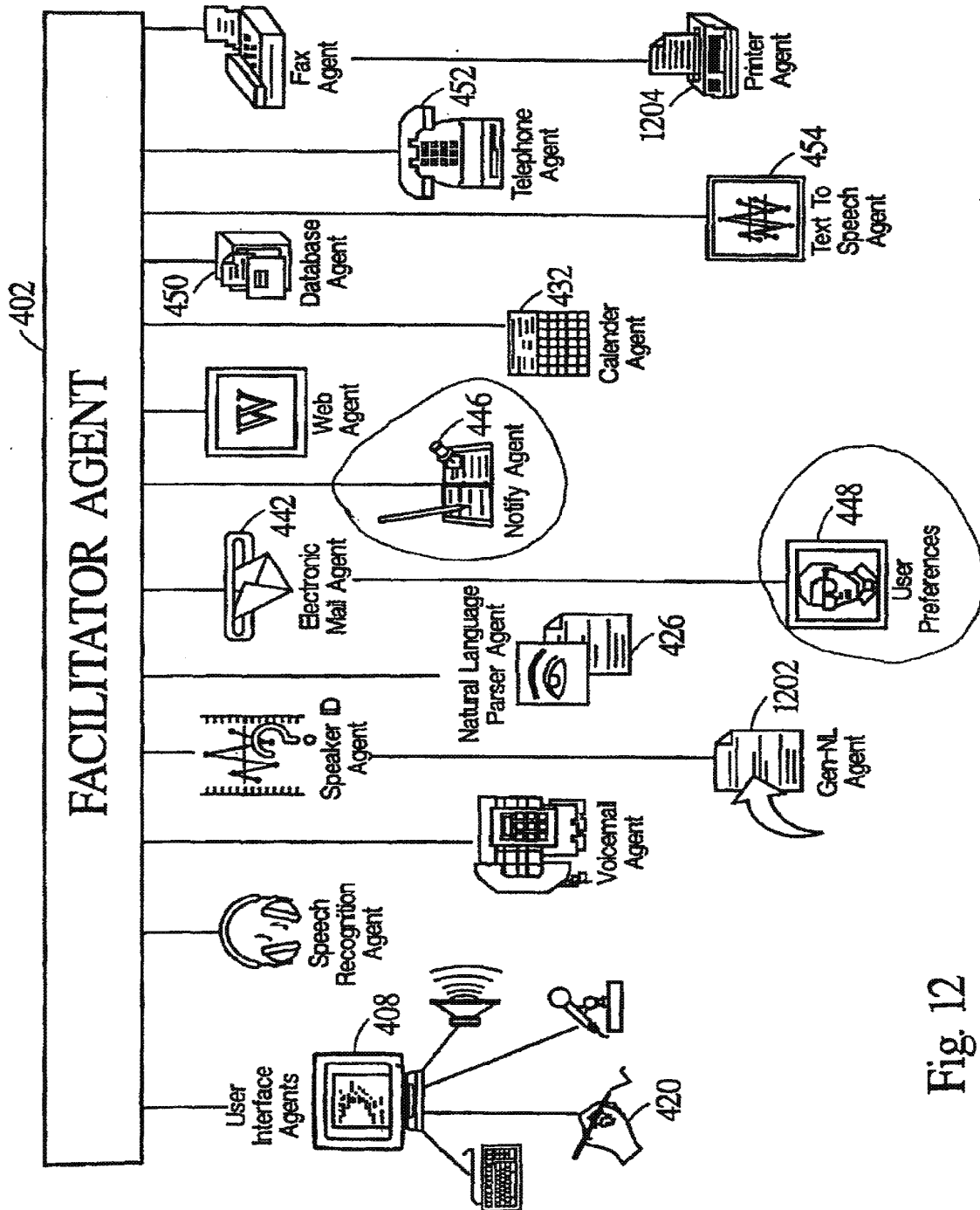


Fig. 12

U.S. Patent

Feb. 1, 2005

Sheet 13 of 16

US 6,851,115 B1

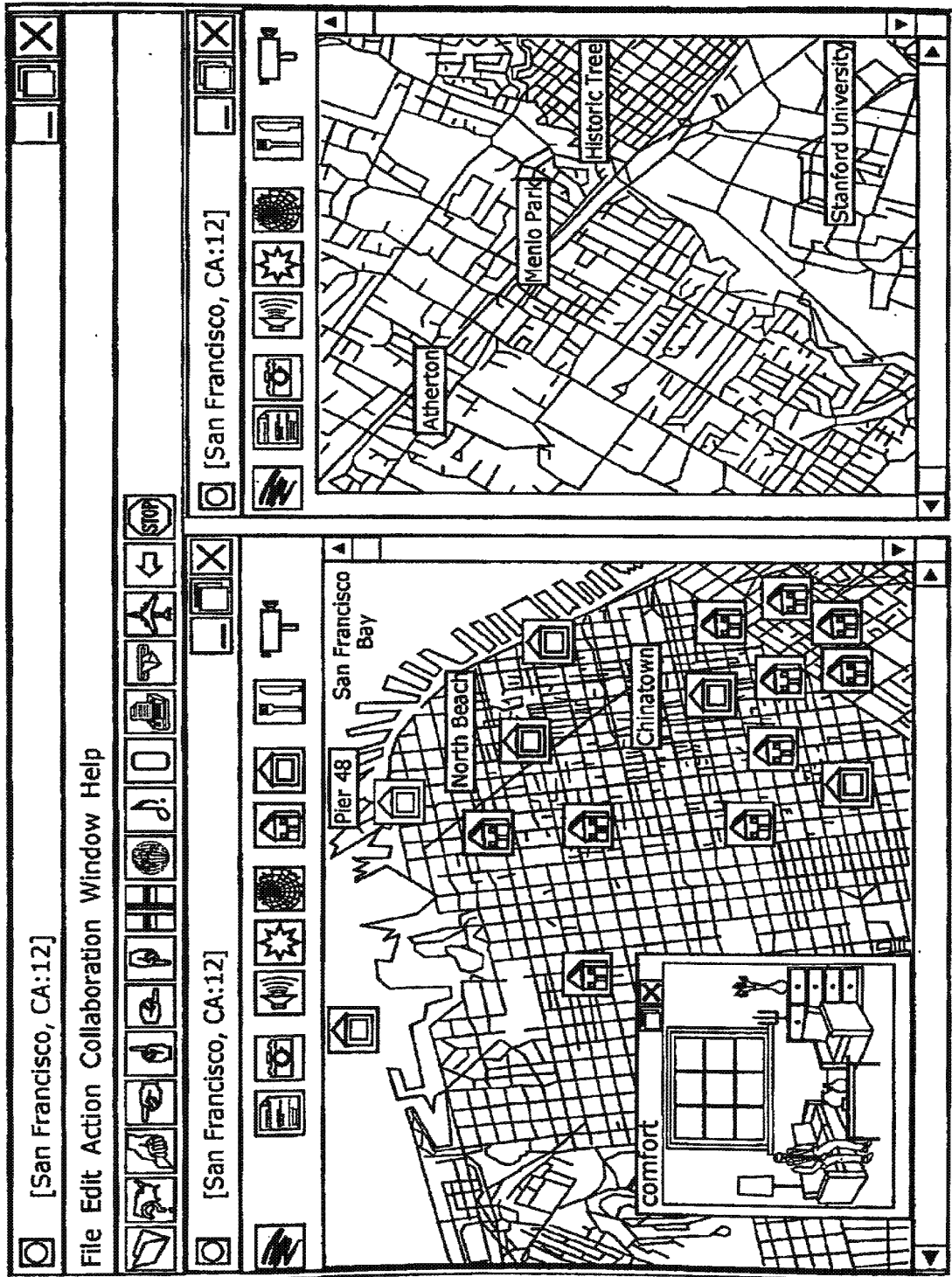


Fig. 13

U.S. Patent

Feb. 1, 2005

Sheet 14 of 16

US 6,851,115 B1

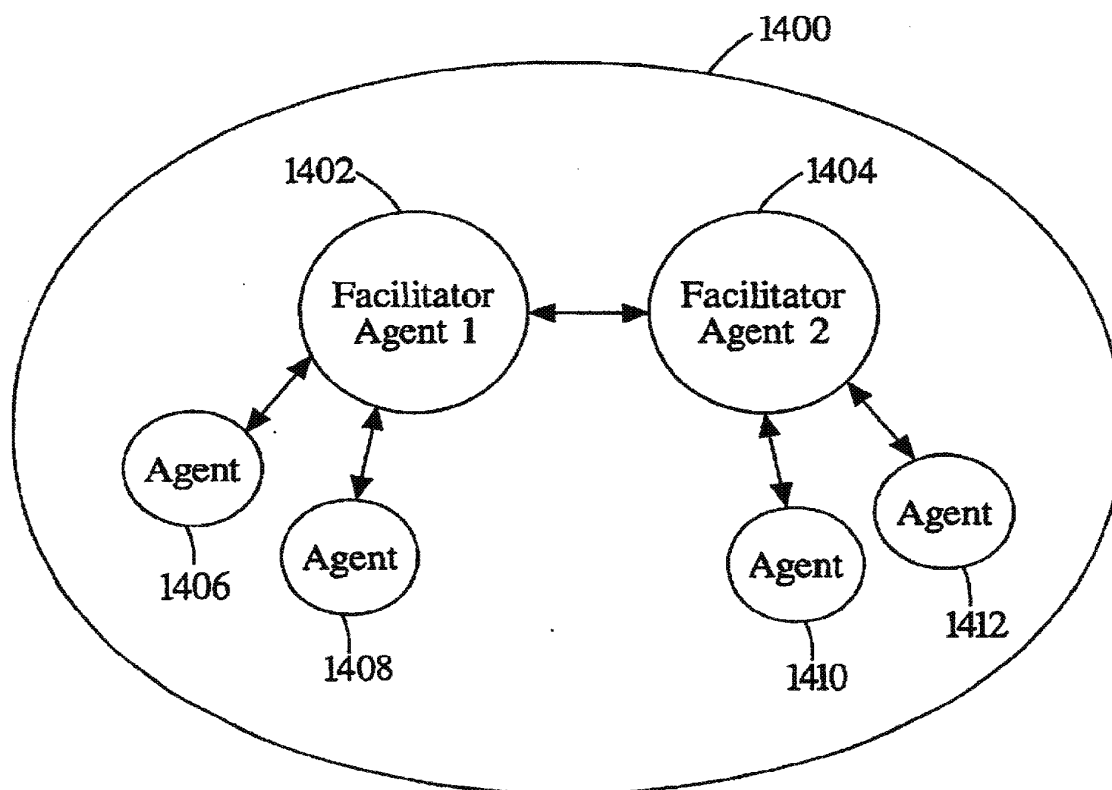


Fig. 14

U.S. Patent

Feb. 1, 2005

Sheet 15 of 16

US 6,851,115 B1

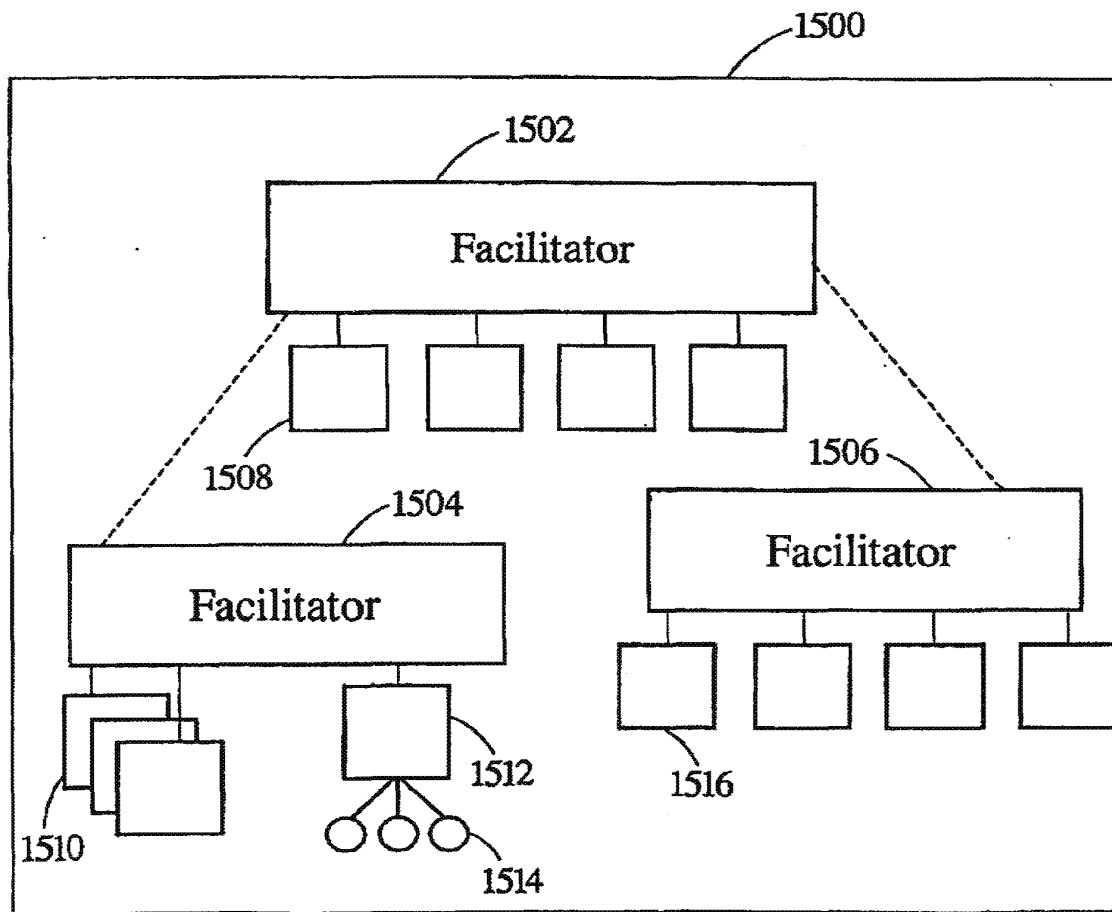


Fig. 15

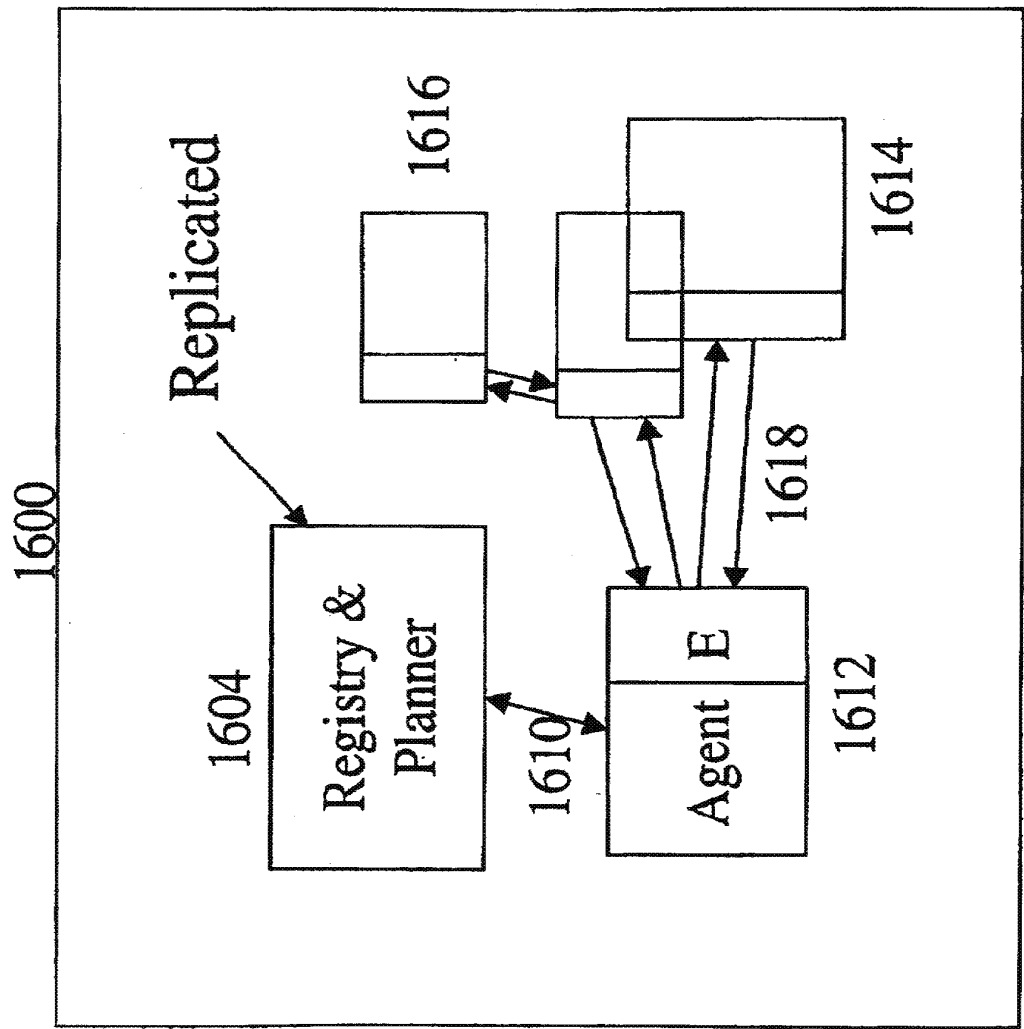


Figure 16

US 6,851,115 B1

1

SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS

A compact disk containing a computer program listing has been provided in duplicate (copy 1 and copy 2 of the compact disk are identical). The computer program listing in the compact disk is incorporated by reference herein. The compact disk contains files with their names, size and date of creation as follow:

File Name	Size	Creation Date	Last Date
oaa.pl	159,613 bytes	1996/10/08	1998/12/23
fac.pl	52,733 bytes	1997/04/24	1998/05/06
compound.pl	42,937 bytes	1996/12/11	1998/04/10
com_tcp.pl	18,010 bytes	1998/02/10	1998/05/06
translations.pl	19,583 bytes	1998/01/29	1998/12/23

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is related to distributed computing environments and the completion of tasks within such environments. In particular, the present invention teaches a variety of software-based architectures for communication and cooperation among distributed electronic agents. Certain embodiments teach interagent communication languages enabling client agents to make requests in the form of arbitrarily complex goal expressions that are solved through facilitation by a facilitator agent.

Context and Motivation for Distributed Software Systems

The evolution of models for the design and construction of distributed software systems is being driven forward by several closely interrelated trends: the adoption of a networked computing model, rapidly rising expectations for smarter, longer-lived, more autonomous software applications and an ever increasing demand for more accessible and intuitive user interfaces.

Prior Art FIG. 1 illustrates a networked computing model 100 having a plurality of client and server computer systems 120 and 122 coupled together over a physical transport mechanism 140. The adoption of the networked computing model 100 has lead to a greatly increased reliance on distributed sites for both data and processing resources. Systems such as the networked computing model 100 are based upon at least one physical transport mechanism 140 coupling the multiple computer systems 120 and 122 to support the transfer of information between these computers.

Some of these computers basically support using the network and are known as client computers (clients). Some of these computers provide resource to other computers and are known as server computers (servers). The servers 122 can vary greatly in the resources they possess, access they provide and services made available to other computers across a network. Servers may service other servers as well as clients.

The Internet is a computing system based upon this network computing model. The Internet is continually growing, stimulating a paradigm shift for computing away from requiring all relevant data and programs to reside on the user's desktop machine. The data now routinely accessed from computers spread around the world has become increasingly rich in format, comprising multimedia

2

documents, and audio and video streams. With the popularization of programming languages such as JAVA, data transported between local and remote machines may also include programs that can be downloaded and executed on the local machine. There is an ever increasing reliance on networked computing, necessitating software design approaches that allow for flexible composition of distributed processing elements in a dynamically changing and relatively unstable environment.

In an increasing variety of domains, application designers and users are coming to expect the deployment of smarter, longer-lived, more autonomous, software applications. Push technology, persistent monitoring of information sources, and the maintenance of user models, allowing for personalized responses and sharing of preferences, are examples of the simplest manifestations of this trend. Commercial enterprises are introducing significantly more advanced approaches, in many cases employing recent research results from artificial intelligence, data mining, machine learning, and other fields.

More than ever before, the increasing complexity of systems, the development of new technologies, and the availability of multimedia material and environments are creating a demand for more accessible and intuitive user interfaces. Autonomous, distributed, multi-component systems providing sophisticated services will no longer lend themselves to the familiar "direct manipulation" model of interaction, in which an individual user masters a fixed selection of commands provided by a single application. Ubiquitous computing, in networked environments, has brought about a situation in which the typical user of many software services is likely to be a non-expert, who may access a given service infrequently or only a few times. Accommodating such usage patterns calls for new approaches, fortunately, input modalities now becoming widely available, such as speech recognition and pen-based handwriting/gesture recognition, and the ability to manage the presentation of systems' responses by using multiple media provide an opportunity to fashion a style of human-computer interaction that draws much more heavily on our experience with human-human interactions.

2. Prior Related Art

Existing approaches and technologies for distributed computing include to distributed objects, mobile objects, blackboard-style architectures, and agent-based software engineering.

The Distributed Object Approach

Object-oriented languages, such as C++ or JAVA, provide significant advances over standard procedural languages with respect to the reusability and modularity of code: encapsulation, inheritance and polymorphism. Encapsulation encourages the creation of library interfaces that minimize dependencies on underlying algorithms or data structures. Changes to programming internals can be made at a later date with requiring modifications to the code that uses the library. Inheritance permits the extension and modification of a library of routines and data without requiring source code to the original library. Polymorphism allows one body of code to work on an arbitrary number of data types. For the sake of simplicity traditional objects may be seen to contain both methods and data. Methods provide the mechanisms by which the internal state of an object may be modified or by which communication may occur with another object or by which the instantiation or removal of objects may be directed.

With reference to FIG. 2, a distributed object technology based around an Object Request Broker will now be

US 6,851,115 B1

3

described. Whereas "standard" object-oriented programming (OOP) languages can be used to build monolithic programs out of many object building blocks, distributed object technologies (DOOP) allow the creation of programs whose components may be spread across multiple machines. As shown in FIG. 2, an object system 200 includes client objects 210 and server objects 220. To implement a client-server relationship between objects, the distributed object system 200 uses a registry mechanism (CORBA's registry is called an object Request Broker, or ORB) 230 to store the interface descriptions of available objects. Through the services of the ORB 230, a client can transparently invoke a method on a remote server object. The ORB 230 is then responsible for finding the object 220 that can implement the request, passing it the parameters, invoking its method, and returning the results. In the most sophisticated systems, the client 210 does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of the server object's interface.

Although distributed objects offer a powerful paradigm for creating networked applications, certain aspects of the approach are not perfectly tailored to the constantly changing environment of the Internet. A major restriction of the DOOP approach is that the interactions among objects are fixed through explicitly coded instructions by the application developer. It is often difficult to reuse an object in a new application without bringing along all its inherent dependencies on other objects (embedded interface definitions and explicit method calls). Another restriction of the DOOP approach is the result of its reliance on a remote procedure call (RPC) style of communication. Although easy to debug, this single thread of execution model does not facilitate programming to exploit the potential for parallel computation that one would expect in a distributed environment. In addition, RPC uses a blocking (synchronous) scheme that does not scale well for high-volume transactions.

Mobile Objects

Mobile objects, sometimes called mobile agents, are bits of code that can move to another execution site (presumably on a different machine) under their own programmatic control, where they can then interact with the local environment. For certain types of problems, the mobile object paradigm offers advantages over more traditional distributed object approaches. These advantages include network bandwidth and parallelism. Network bandwidth advantages exist for some database queries or electronic commerce applications, where it is more efficient to perform tests on data by bringing the tests to the data than by bringing large amounts of data to the testing program. Parallelism advantages include situations in which mobile agents can be spawned in parallel to accomplish many tasks at once.

Some of the disadvantages and inconveniences of the mobile agent approach include the programmatic specificity of the agent interactions, lack of coordination support between participant agents and execution environment irregularities regarding specific programming languages supported by host processors upon which agents reside. In a fashion similar to that of DOOP programming, an agent developer must programmatically specify where to go and how to interact with the target environment. There is generally little coordination support to encourage interactions among multiple (mobile) participants. Agents must be written in the programming language supported by the execution environment, whereas many other distributed technologies support heterogeneous communities of components, written in diverse programming languages.

4

Blackboard Architectures

Blackboard architectures typically allow multiple processes to communicate by reading and writing tuples from a global data store. Each process can watch for items of interest, perform computations based on the state of the blackboard, and then add partial results or queries that other processes can consider. Blackboard architectures provide a flexible framework for problem solving by a dynamic community of distributed processes. A blackboard architecture provides one solution to eliminating the tightly bound interaction links that some of the other distributed technologies require during interprocess communication. This advantage can also be a disadvantage: although a programmer does not need to refer to a specific process during computation, the framework does not provide programmatic control for doing so in cases where this would be practical.

Agent-based Software Engineering

Several research communities have approached distributed computing by casting it as a problem of modeling communication and cooperation among autonomous entities, or agents. Effective communication among independent agents requires four components: (1) a transport mechanism carrying messages in an asynchronous fashion, (2) an interaction protocol defining various types of communication interchange and their social implications (for instance, a response is expected of a question), (3) a content language permitting the expression and interpretation of utterances, and (4) an agreed-upon set of shared vocabulary and meaning for concepts often called an ontology). Such mechanisms permit a much richer style of interaction among participants than can be expressed using a distributed object's RPC model or a blackboard architecture's centralized exchange approach.

Agent-based systems have shown much promise for flexible, fault-tolerant, distributed problem solving. Several agent-based projects have helped to evolve the notion of facilitation. However, existing agent-based technologies and architectures are typically very limited in the extent to which agents can specify complex goals or influence the strategies used by the facilitator. Further, such prior systems are not sufficiently attuned to the importance of integrating human agents (i.e., users) through natural language and other human-oriented user interface technologies.

The initial version of SRI International's Open Agent Architecture™ ("OAA®") technology provided only a very limited mechanism for dealing with compound goals. Fixed formats were available for specifying a flat list of either conjoined (AND) sub-goals or disjointed (OR) sub-goals; in both cases, parallel goal solving was hard-wired in, and only a single set of parameters for the entire list could be specified. More complex goal expressions involving (for example) combinations of different boolean connectors, nested expressions, or conditionally interdependent ("IF... THEN") goals were not supported. Further, system scalability was not adequately addressed in this prior work.

SUMMARY OF INVENTION

A first embodiment of the present invention discloses a highly flexible, software-based architecture for constructing distributed systems. The architecture supports cooperative task completion by flexible, dynamic configurations of autonomous electronic agents. Communication and cooperation between agents are brokered by one or more facilitators, which are responsible for matching requests, from users and agents, with descriptions of the capabilities of other agents. It is not generally required that a user or agent know the identities, locations, or number of other

US 6,851,115 B1

5

agents involved in satisfying a request, and relatively minimal effort is involved in incorporating new agents and “wrapping” legacy applications. Extreme flexibility is achieved through an architecture organized around the declaration of capabilities by service-providing agents, the construction of arbitrarily complex goals by users and service-requesting agents, and the role of facilitators in delegating and coordinating the satisfaction of these goals, subject to advice and constraints that may accompany them. Additional mechanisms and features include facilities for creating and maintaining shared repositories of data; the use of triggers to instantiate commitments within and between agents; agent-based provision of multi-modal user interfaces, including natural language; and built-in support for including the user as a privileged member of the agent community. Specific embodiments providing enhanced scalability are also described.

BRIEF DESCRIPTION OF THE DRAWINGS

Prior Art

Prior Art FIG. 1 depicts a networked computing model;

Prior Art FIG. 2 depicts a distributed object technology based around an Object Resource Broker;

Examples of the Invention

FIG. 3 depicts a distributed agent system based around a facilitator agent;

FIG. 4 presents a structure typical of one small system of the present invention;

FIG. 5 depicts an Automated Office system implemented in accordance with an example embodiment of the present invention supporting a mobile user with a laptop computer and a telephone;

FIG. 6 schematically depicts an Automated Office system implemented as a network of agents in accordance with a preferred embodiment of the present invention;

FIG. 7 schematically shows data structures internal to a facilitator in accordance with a preferred embodiment of the present invention;

FIG. 8 depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention;

FIG. 9 depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a certain preferred embodiment of the present invention;

FIG. 10 depicts operations involved in a client agent responding to a service request in accordance with another preferable embodiment of the present invention;

FIG. 11 depicts operations involved in a facilitator agent response to a service request in accordance with a preferred embodiment of the present invention;

FIG. 12 depicts an Open Agent Architecture™ based system of agents implementing a unified messaging application in accordance with a preferred embodiment of the present invention;

FIG. 13 depicts a map oriented graphical user interface display as might be displayed by a multi-modal map application in accordance with a preferred embodiment of the present invention;

FIG. 14 depicts a peer to peer multiple facilitator based agent system supporting distributed agents in accordance with a preferred embodiment of the present invention;

FIG. 15 depicts a multiple facilitator agent system supporting at least a limited form of a hierarchy of facilitators in accordance with a preferred embodiment of the present invention; and

FIG. 16 depicts a replicated facilitator architecture in accordance with one embodiment of the present invention.

6

DETAILED DESCRIPTION OF THE INVENTION

FIG. 3 illustrates a distributed agent system 300 in accordance with one embodiment of the present invention. The agent system 300 includes a facilitator agent 310 and a plurality of agents 320. The illustration of FIG. 3 provides a high level view of one simple system structure contemplated by the present invention. The facilitator agent 310 is in essence the “parent” facilitator for its “children” agents 320. The agents 320 forward service requests to the facilitator agent 310. The facilitator agent 310 interprets these requests, organizing a set of goals which are then delegated to appropriate agents for task completion.

The system 300 of FIG. 3 can be expanded upon and modified in a variety of ways consistent with the present invention. For example, the agent system 300 can be distributed across a computer network such as that illustrated in FIG. 1. The facilitator agent 310 may itself have its functionality distributed across several different computing platforms. The agents 320 may engage in interagent communication (also called peer to peer communications). Several different systems 300 may be coupled together for enhanced performance. These and a variety of other structural configurations are described below in greater detail.

FIG. 4 presents the structure typical of a small system 400 in one embodiment of the present invention, showing user interface agents 408, several application agents 404 and meta-agents 406, the system 400 organized as a community of peers by their common relationship to a facilitator agent 402. As will be appreciated, FIG. 4 places more structure upon the system 400 than shown in FIG. 3, but both are valid representations of structures of the present invention. The facilitator 402 is a specialized server agent that is responsible for coordinating agent communications and cooperative problem-solving. The facilitator 402 may also provide a global data store for its client agents, allowing them to adopt a blackboard style of interaction. Note that certain advantages are found in utilizing two or more facilitator agents within the system 400. For example, larger systems can be assembled from multiple facilitator/client groups, each having the sort of structure shown in FIG. 4. All agents that are not facilitators are referred to herein generically as client agents—so called because each acts (in some respects) as a client of some facilitator, which provides communication and other essential services for the client.

The variety of possible client agents is essentially unlimited. Some typical categories of client agents would include application agents 404, meta-agents 406, and user interface agents 408, as depicted in FIG. 4. Application agents 404 denote specialists that provide a collection of services of a particular sort. These services could be domain-independent technologies (such as speech recognition, natural language processing 410, email, and some forms of data retrieval and data mining) or user-specific or domain-specific (such as a travel planning and reservations agent). Application agents may be based on legacy applications or libraries, in which case the agent may be little more than a wrapper that calls a pre-existing API 412, for example. Meta-agents 406 are agents whose role is to assist the facilitator agent 402 in coordinating the activities of other agents. While the facilitator 402 possesses domain-independent coordination strategies, meta-agents 406 can augment these by using domain- and application-specific knowledge or reasoning (including but not limited to rules, learning algorithms and planning).

With further reference to FIG. 4, user interface agents 408 can play an extremely important and interesting role in

US 6,851,115 B1

7

certain embodiments of the present invention. By way of explanation, in some systems, a user interface agent can be implemented as a collection of "micro-agents", each monitoring a different input modality (point-and-click, handwriting, pen gestures, speech), and collaborating to produce the best interpretation of the current inputs. These micro-agents are depicted in FIG. 4, for example, as Modality Agents 414. While describing such subcategories of client agents is useful for purposes of illustration and understanding, they need not be formally distinguished within the system in preferred implementations of the present invention.

The operation of one preferred embodiment of the present invention will be discussed in greater detail below, but may be briefly outlined as follows. When invoked, a client agent makes a connection to a facilitator, which is known as its parent facilitator. These connections are depicted as a double headed arrow between the client agent and the facilitator agent in FIGS. 3 and 4, for example. Upon connection, an agent registers with its parent facilitator a specification of the capabilities and services it can provide. For example, a natural language agent may register the characteristics of its available natural language vocabulary. (For more details regarding client agent connections, see the discussion of FIG. 8 below.) Later during task completion, when a facilitator determines that the registered services 416 of one of its client agents will help satisfy a goal, the facilitator sends that client a request expressed in the Interagent Communication Language (ICL) 418. (See FIG. 11 below for a more detailed discussion of the facilitator operations involved.) The agent parses this request, processes it, and returns answers or status reports to the facilitator. In processing a request, the client agent can make use of a variety of infrastructure capabilities provided in the preferred embodiment. For example, the client agent can use ICL 418 to request services of other agents, set triggers, and read or write shared data on the facilitator or other client agents that maintain shared data. (See the discussion of FIGS. 9-11 below for a more detailed discussion of request processing.)

The functionality of each client agent are made available to the agent community through registration of the client agent's capabilities with a facilitator 402. A software "wrapper" essentially surrounds the underlying application program performing the services offered by each client. The common infrastructure for constructing agents is preferably supplied by an agent library. The agent library is preferably accessible in the runtime environment of several different programming languages. The agent library preferably minimizes the effort required to construct a new system and maximizes the ease with which legacy systems can be "wrapped" and made compatible with the agent-based architecture of the present invention.

By way of further illustration, a representative application is now briefly presented with reference to FIGS. 5 and 6. In the Automated Office system depicted in FIG. 5, a mobile user with a telephone and a laptop computer can access and task commercial applications such as calendars, databases, and email systems running back at the office. A user interface (UI) agent 408, shown in FIG. 6, runs on the user's local laptop and is responsible for accepting user input, sending requests to the facilitator 402 for delegation to appropriate agents, and displaying the results of the distributed computation. The user may interact directly with a specific remote application by clicking on active areas in the interface, calling up a form or window for that application, and making queries with standard interface dialog mechanisms. Conversely, a user may express a task to be executed by

8

using typed, handwritten, or spoken (over the telephone) English sentences, without explicitly specifying which agent or agents should perform the task.

For instance, if the question "What is my schedule?" is written 420 in the user interface 408, this request will be sent 422 by the UI 408 to the facilitator 402, which in turn will ask 424 a natural language (NL) agent 426 to translate the query into JCL 18. To accomplish this task, the NL agent 426 may itself need to make requests of the agent community to resolve unknown words such as "me" 428 (the UI agent 408 can respond 430 with the name of the current user) or "schedule" 432 (the calendar agent 434 defines this word 436). The resulting ICL expression is then routed by the facilitator 402 to appropriate agents (in this case, the calendar agent 434) to execute the request. Results are sent back 438 to the UI agent 408 for display.

The spoken request "When mail arrives for me about security, notify me immediately." produces a slightly more complex example involving communication among all agents in the system. After translation into ICL as described above, the facilitator installs a trigger 440 on the mail agent 442 to look for new messages about security. When one such message does arrive in its mail spool, the trigger fires, and the facilitator matches the action part of the trigger to capabilities published by the notification agent 446. The notification agent 446 is a meta-agent, as it makes use of rules concerning the optimal use of different output modalities (email, fax, speech generation over the telephone) plus information about an individual user's preferences 448 to determine the best way of relaying a message through available media transfer application agents. After some competitive parallelism to locate the user (the calendar agent 434 and database agent 450 may have different guesses as to where to find the user) and some cooperative parallelism to produce required information (telephone number of location, user password, and an audio file containing a text-to-speech representation of the email message), a telephone agent 452 calls the user, verifying its identity through touchtones, and then play the message.

The above example illustrates a number of inventive features. As new agents connect to the facilitator, registering capability specifications and natural language vocabulary, what the user can say and do dynamically changes; in other words, the ICL is dynamically expandable. For example, adding a calendar agent to the system in the previous example and registering its capabilities enables users to ask natural language questions about their "schedule" without any need to revise code for the facilitator, the natural language agents, or any other client agents. In addition, the interpretation and execution of a task is a distributed process, with no single agent defining the set of possible inputs to the system. Further, a single request can produce cooperation and flexible communication among many agents, written in different programming languages and spread across multiple machines.

Design Philosophy and Considerations

One preferred embodiment provides an integration mechanism for heterogeneous applications in a distributed infrastructure, incorporating some of the dynamism and extensibility of blackboard approaches, the efficiency associated with mobile objects, plus the rich and complex interactions of communicating agents. Design goals for preferred embodiments of the present invention may be categorized under the general headings of interoperation and cooperation, user interfaces, and software engineering. These design goals are not absolute requirements, nor will they necessarily be satisfied by all embodiments of the

US 6,851,115 B1

9

present invention, but rather simply reflect the inventor's currently preferred design philosophy.

Versatile Mechanisms of Interoperation and Cooperation

Interoperation refers to the ability of distributed software components—agents—to communicate meaningfully. While every system-building framework must provide mechanisms of interoperation at some level of granularity, agent-based frameworks face important new challenges in this area. This is true primarily because autonomy, the hallmark of individual agents, necessitates greater flexibility in interactions within communities of agents. Coordination refers to the mechanisms by which a community of agents is able to work together productively on some task. In these areas, the goals for our framework are to provide flexibility in assembling communities of autonomous service providers, provide flexibility in structuring cooperative interactions, impose the right amount of structure, as well as include legacy and “owned-elsewhere” applications.

Provide flexibility in assembling communities of autonomous service providers—both at development time and at runtime. Agents that conform to the linguistic and ontological requirements for effective communication should be able to participate in an agent community, in various combinations, with minimal or near minimal prerequisite knowledge of the characteristics of the other players. Agents with duplicate and overlapping capabilities should be able to coexist within the same community, with the system making optimal or near optimal use of the redundancy.

Provide flexibility in structuring cooperative interactions among the members of a community of agents. A framework preferably provides an economical mechanism for setting up a variety of interaction patterns among agents, without requiring an inordinate amount of complexity or infrastructure within the individual agents. The provision of a service should be independent or minimally dependent upon a particular configuration of agents.

Impose the right amount of structure on individual agents. Different approaches to the construction of multi-agent systems impose different requirements on the individual agents. For example, because KQML is neutral as to the content of messages, it imposes minimal structural requirements on individual agents. On the other hand, the BDI paradigm tends to impose much more demanding requirements, by making assumptions about the nature of the programming elements that are meaningful to individual agents. Preferred embodiments of the present invention should fall somewhere between the two, providing a rich set of interoperation and coordination capabilities, without precluding any of the software engineering goals defined below.

Include legacy and “owned-elsewhere” applications. Whereas legacy usually implies reuse of an established system fully controlled by the agent-based system developer, owned-elsewhere refers to applications to which the developer has partial access, but no control. Examples of owned-elsewhere applications include data sources and services available on the World Wide Web, via simple form-based interfaces, and applications used cooperatively within a virtual enterprise, which remain the properties of separate corporate entities. Both classes of application must preferably be able to interoperate, more or less as full-fledged members of the agent community, without requiring an overwhelming integration effort.

Human-Oriented User Interfaces

Systems composed of multiple distributed components, and possibly dynamic configurations of components, require the crafting of intuitive user interfaces to provide conceptually natural interaction mechanisms, treat users as privileged members of the agent community and support collaboration.

10

Provide conceptually natural interaction mechanisms with multiple distributed components. When there are numerous disparate agents, and/or complex tasks implemented by the system, the user should be able to express requests without having detailed knowledge of the individual agents. With speech recognition, handwriting recognition, and natural language technologies becoming more mature, agent architectures should preferably support these forms of input playing increased roles in the tasking of agent communities.

Preferably treat users as privileged members of the agent community by providing an appropriate level of task specification within software agents, and reusable translation mechanisms between this level and the level of human requests, supporting constructs that seamlessly incorporate interactions between both human-interface and software types of agents.

Preferably support collaboration (simultaneous work over shared data and processing resources) between users and agents.

Realistic Software Engineering Requirements

System-building frameworks should preferably address the practical concerns of real-world applications by the specification of requirements which preferably include: Minimize the effort required to create new agents, and to wrap existing applications. Encourage reuse, both of domain-independent and domain-specific components. The concept of agent orientation, like that of object orientation, provides a natural conceptual framework for reuse, so long as mechanisms for encapsulation and interaction are structured appropriately. Support lightweight mobile platforms. Such platforms should be able to serve as hosts for agents, without requiring the installation of a massive environment. It should also be possible to construct individual agents that are relatively small and modest in their processing requirements. Minimize platform and language barriers. Creation of new agents, as well as wrapping of existing applications, should not require the adoption of a new language or environment.

Mechanisms of Cooperation

Cooperation among agents in accordance with the present invention is preferably achieved via messages expressed in a common language, ICL. Cooperation among agent is further preferably structured around a three-part approach: providers of services register capabilities specifications with a facilitator, requesters of services construct goals and relay them to a facilitator, and facilitators coordinate the efforts of the appropriate service providers in satisfying these goals. The Interagent Communication Language (ICL)

Interagent Communication Language (“ICL”) 418 refers to an interface, communication, and task coordination language preferably shared by all agents, regardless of what platform they run on or what computer language they are programmed in. ICL may be used by an agent to task itself or some subset of the agent community. Preferably, ICL allows agents to specify explicit control parameters while simultaneously supporting expression of goals in an underspecified, loosely constrained manner. In a further preferred embodiment, agents employ ICL to perform queries, execute actions, exchange information, set triggers, and manipulate data in the agent community.

In a further preferred embodiment, a program element expressed in ICL is the event. The activities of every agent, as well as communications between agents, are preferably structured around the transmission and handling of events. In communications, events preferably serve as messages between agents; in regulating the activities of individual agents, they may preferably be thought of as goals to be

US 6,851,115 B1

11

satisfied. Each event preferably has a type, a set of parameters, and content. For example, the agent library procedure `oaa_Solve` can be used by an agent to request services of other agents. A call to `oaa_Solve`, within the code of agent A, results in an event having the form

`ev_post_solve(Goal, Params)`

going from A to the facilitator, where `ev_post_solve` is the type, `Goal` is the content, and `Params` is a list of parameters. The allowable content and parameters preferably vary according to the type of the event.

The ICL preferably includes a layer of conversational protocol and a content layer. The conversational layer of ICL is defined by the event types, together with the parameter lists associated with certain of these event types. The content layer consists of the specific goals, triggers, and data elements that may be embedded within various events.

The ICL conversational protocol is preferably specified using an orthogonal, parameterized approach, where the conversational aspects of each element of an interagent conversation are represented by a selection of an event type and a selection of values from at least one orthogonal set of parameters. This approach offers greater expressiveness than an approach based solely on a fixed selection of speech acts, such as embodied in KQML. For example, in KQML, a request to satisfy a query can employ either of the performatives `ask_all` or `ask_one`. In ICL, on the other hand, this type of request preferably is expressed by the event type `evost_solve`, together with the `solution_limit(N)` parameter—where N can be any positive integer. (A request for all solutions is indicated by the omission of the `solution_limit` parameter.) The request can also be accompanied by other parameters, which combine to further refine its semantics. In KQML, then, this example forces one to choose between two possible conversational options, neither of which may be precisely what is desired. In either case, the performative chosen is a single value that must capture the entire conversational characterization of the communication. This requirement raises a difficult challenge for the language designer, to select a set of performatives that provides the desired functionality without becoming unmanageably large. Consequently, the debate over the right set of performatives has consumed much discussion within the KQML community.

The content layer of the ICL preferably supports unification and other features found in logic programming language environments such as PROLOG. In some embodiments, the content layer of the ICL is simply an extension of at least one programming language. For example, the Applicants have found that PROLOG is suitable for implementing and extending into the content layer of the ICL. The agent libraries preferably provide support for constructing, parsing, and manipulating ICL expressions. It is possible to embed content expressed in other languages within an ICL event. However, expressing content in ICL simplifies the facilitator's access to the content, as well as the conversational layer, in delegating requests. This gives the facilitator more information about the nature of a request and helps the facilitator decompose compound requests and delegate the sub-requests.

Further, ICL expressions preferably include, in addition to events, at least one of the following: capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements. A further preferred embodiment of the present invention incorporates ICL expressions including at least all of the following: events, capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements.

12

Providing Services: Specifying "Solvables"

In a preferred embodiment of the present invention, every participating agent defines and publishes a set of capability declarations, expressed in ICL, describing the services that it provides. These declarations establish a high-level interface to the agent. This interface is used by a facilitator in communicating with the agent, and, most important, in delegating service requests (or parts of requests) to the agent. Partly due to the use of PROLOG as a preferred basis for ICL, these capability declarations are referred to as solvables. The agent library preferably provides a set of procedures allowing an agent to add, remove, and modify its solvables, which it may preferably do at any time after connecting to its facilitator.

There are preferably at least two major types of solvables: procedure solvables and data solvables. Intuitively, a procedure solvable performs a test or action, whereas a data solvable provides access to a collection of data. For example, in creating an agent for a mail system, procedure solvables might be defined for sending a message to a person, testing whether a message about a particular subject has arrived in the mail queue, or displaying a particular message onscreen. For a database wrapper agent, one might define a distinct data solvable corresponding to each of the relations present in the database. Often, a data solvable is used to provide a shared data store, which may be not only queried, but also updated, by various agents having the required permissions.

There are several primary technical differences between these two types of solvables. First, each procedure solvable must have a handler declared and defined for it, whereas this is preferably not necessary for a data solvable. The handling of requests for a data solvable is preferably provided transparently by the agent library. Second, data solvables are preferably associated with a dynamic collection of facts (or clauses), which may be further preferably modified at runtime, both by the agent providing the solvable, and by other agents (provided they have the required permissions). Third, special features, available for use with data solvables, preferably facilitate maintaining the associated facts. In spite of these differences, it should be noted that the mechanism of use by which an agent requests a service is the same for the two types of solvables.

In one embodiment, a request for one of an agent's services normally arrives in the form of an event from the agent's facilitator. The appropriate handler then deals with this event. The handler may be coded in whatever fashion is most appropriate, depending on the nature of the task, and the availability of task-specific libraries or legacy code, if any. The only hard requirement is that the handler return an appropriate response to the request, expressed in ICL. Depending on the nature of the request, this response could be an indication of success or failure, or a list of solutions (when the request is a data query).

A solvable preferably has three parts: a goal, a list of parameters, and a list of permissions, which are declared using the format:

`solvable(Goal, Parameters, Permissions)`

The goal of a solvable, which syntactically takes the preferable form of an ICL structure, is a logical representation of the service provided by the solvable. (An ICL structure consists of a functor with 0 or more arguments. For example, in the structure `a(b,c)`, 'a' is the functor, and 'b' and 'c' the arguments.) As with a PROLOG structure, the goal's arguments themselves may preferably be structures.

Various options can be included in the parameter list, to refine the semantics associated with the solvable. The type

US 6,851,115 B1

13

parameter is preferably used to say whether the solvable is data or procedure. When the type is procedure, another parameter may be used to indicate the handler to be associated with the solvable. Some of the parameters appropriate for a data solvable are mentioned elsewhere in this application. In either case (procedure or data solvable), the private parameter may be preferably used to restrict the use of a solvable to the declaring agent when the agent intends the solvable to be solely for its internal use but wishes to take advantage of the mechanisms in accordance with the present invention to access it, or when the agent wants the solvable to be available to outside agents only at selected times. In support of the latter case, it is preferable for the agent to change the status of a solvable from private to non-private at any time.

The permissions of a solvable provide mechanisms by which an agent may preferably control access to its services allowing the agent to restrict calling and writing of a solvable to itself and/or other selected agents. (Calling means requesting the service encapsulated by a solvable, whereas Writing means modifying the collection of facts associated with a data solvable.) The default permission for every solvable in a further preferred embodiment of the present invention is to be callable by anyone, and for data solvables to be writable by anyone. A solvable's permissions can preferably be changed at any time, by the agent providing the solvable.

For example, the solvables of a simple email agent might include:

```
solvable(send_message(email, +ToPerson, +Params),
  [type(procedure), callback(send_mail)],
  [ ])
solvable(last_message(email, -MessageId),
  [type(data), single_value(true)],
  [write(true)]),
solvable(get_message(email, +MessageId, -Msg),
  [type(procedure), callback(get_mail)], [ ])
```

The symbols '+' and '-', indicating input and output arguments, are at present used only for purposes of documentation. Most parameters and permissions have default values, and specifications of default values may be omitted from the parameters and permissions lists.

Defining an agent's capabilities in terms of solvable declarations effectively creates a vocabulary with which other agents can communicate with the new agent. Ensuring that agents will speak the same language and share a common, unambiguous semantics of the vocabulary involves ontology. Agent development tools and services (automatic translations of solvables by the facilitator) help address this issue; additionally, a preferred embodiment of the present invention will typically rely on vocabulary from either formally engineered ontologies for specific domains or from ontologies constructed during the incremental development of a body of agents for several applications or from both specific domain ontologies and incrementally developed ontologies. Several example tools and services are described in Cheyer et al.'s paper entitled "Development Tools for the Open Agent Architecture," as presented at the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 96), London, April 1996.

Although the present invention imposes no hard restrictions on the form of solvable declarations, two common usage conventions illustrate some of the utility associated with solvables.

Classes of services are often preferably tagged by a particular type. For instance, in the example above, the "last_message" and "get_message" solvables are special-

14

ized for email, not by modifying the names of the services, but rather by the use of the 'email' parameter, which serves during the execution of an ICL request to select (or not) a specific type of message.

Actions are generally written using an imperative verb as the functor of the solvable in a preferred embodiment of the present invention, the direct object (or item class) as the first argument of the predicate, required arguments following, and then an extensible parameter list as the last argument. The parameter list can hold optional information usable by the function. The ICL expression generated by a natural language parser often makes use of this parameter list to store prepositional phrases and adjectives.

As an illustration of the above two points, "Send mail to Bob about lunch" will be translated into an ICL request send_message(email, 'Bob Jones', [subject(lunch)]), whereas "Remind Bob about lunch" would leave the transport unspecified (send_message(KIND, 'Bob Jones', [subject(lunch)])), enabling an available message transfer agents (e.g., fax, phone, mail, pager) to compete for the opportunity to carry out the request.

Requesting Services

An agent preferably requests services of the community of agent by delegating tasks or goals to its facilitator. Each request preferably contains calls to one or more agent solvables, and optionally specifies parameters containing advice to help the facilitator determine how to execute the task. Calling a solvable preferably does not require that the agent specify (or even know of) a particular agent or agents to handle the call. While it is possible to specify one or more agents using an address parameter (and there are situations in which this is desirable), in general it is advantageous to leave this delegation to the facilitator. This greatly reduces the hard-coded component dependencies often found in other distributed frameworks. The agent libraries of a preferred embodiment of the present invention provide an agent with a single, unified point of entry for requesting services of other agents: the library procedure oaa_Solve. In the style of logic programming, oaa_Solve may preferably be used both to retrieve data and to initiate actions, so that calling a data solvable looks the same as calling a procedure solvable.

Complex Goal Expressions

A powerful feature provided by preferred embodiments of the present invention is the ability of a client agent (or a user) to submit compound goals of an arbitrarily complex nature to a facilitator. A compound goal is a single goal expression that specifies multiple sub-goals to be performed. In speaking of a "complex goal expression" we mean that a single goal expression that expresses multiple sub-goals can potentially include more than one type of logical connector (e.g., AND, OR, NOT), and/or more than one level of logical nesting (e.g., use of parentheses), or the substantive equivalent. By way of further clarification, we note that when speaking of an "arbitrarily complex goal expression" we mean that goals are expressed in a language or syntax that allows expression of such complex goals when appropriate or when desired, not that every goal is itself necessarily complex.

It is contemplated that this ability is provided through an interagent communication language having the necessary syntax and semantics. In one example, the goals may take the form of compound goal expressions composed using operators similar to those employed by PROLOG, that is, the comma for conjunction, the semicolon for disjunction, the arrow for conditional execution, etc. The present invention also contemplates significant extensions to PROLOG

US 6,851,115 B1

15

syntax and semantics. For example, one embodiment incorporates a "parallel disjunction" operator indicating that the disjuncts are to be executed by different agents concurrently. A further embodiment supports the specification of whether a given sub-goal is to be executed breadth-first or depth-first.

A further embodiment supports each sub-goal of a compound goal optionally having an address and/or a set of parameters attached to it. Thus, each sub-goal takes the form

Address:Goal:Parameters

where both Address and Parameters are optional.

An address, if present, preferably specifies one or more agents to handle the given goal, and may employ several different types of referring expression: unique names, symbolic names, and shorthand names. Every agent has preferably a unique name, assigned by its facilitator, which relies upon network addressing schemes to ensure its global uniqueness. Preferably, agents also have self-selected symbolic names (for example, "mail"), which are not guaranteed to be unique. When an address includes a symbolic name, the facilitator preferably takes this to mean that all agents having that name should be called upon. Shorthand names include 'self' and 'parent' (which refers to the agent's facilitator). The address associated with a goal or sub-goal is preferably always optional. When an address is not present, it is the facilitator's job to supply an appropriate address.

The distributed execution of compound goals becomes particularly powerful when used in conjunction with natural language or speech-enabled interfaces, as the query itself may specify how functionality from distinct agents will be combined. As a simple example, the spoken utterance "Fax it to Bill Smith's manager." can be translated into the following compound ICL request:

```
oaa Solve((manager('Bill Smith', M), fax(it,M,[ ])),
[strategy(action)])
```

Note that in this ICL request there are two sub-goals, "manager('Bill Smith',M)" and "fax(it,M,[])," and a single global parameter "strategy(action)." According to the present invention, the facilitator is capable of mapping global parameters in order to apply the constraints or advice across the separate sub-goals in a meaningful way. In this instance, the global parameter strategy(action) implies a parallel constraint upon the first sub-goal; i.e., when there are multiple agents that can respond to the manager sub-goal, each agent should receive a request for service. In contrast, for the second sub-goal, parallelism should not be inferred from the global parameter strategy(action) because such an inference would possibly result in the transmission of duplicate facsimiles.

Refining Service Requests

In a preferred embodiment of the present invention, parameters associated with a goal (or sub-goal) can draw on useful features to refine the request's meaning. For example, it is frequently preferred to be able to specify whether or not solutions are to be returned synchronously; this is done using the reply parameter, which can take any of the values synchronous, asynchronous, or none. As another example, when the goal is a non-compound query of a data solvable, the cache parameter may preferably be used to request local caching of the facts associated with that solvable.

Many of the remaining parameters fall into two categories: feedback and advice. Feedback parameters allow a service requester to receive information from the facilitator about how a goal was handled. This feedback can include such things as the identities of the agents involved in satisfying the goal, and the amount of time expended in the satisfaction of the goal.

Advice parameters preferably give constraints or guidance to the facilitator in completing and interpreting the

16

goal. For example, a solution_limit parameter preferably allows the requester to say how many solutions it is interested in; the facilitator and/or service providers are free to use this information in optimizing their efforts. Similarly, a time_limit is preferably used to say how long the requester is willing to wait for solutions to its request, and, in a multiple facilitator system, a level_limit may preferably be used to say how remote the facilitators may be that are consulted in the search for solutions. A priority parameter is preferably used to indicate that a request is more urgent than previous requests that have not yet been satisfied. Other preferred advice parameters include but are not limited to parameters used to tell the facilitator whether parallel satisfaction of the parts of a goal is appropriate, how to combine and filter results arriving from multiple solver agents, and whether the requester itself may be considered a candidate solver of the sub-goals of a request.

Advice parameters preferably provide an extensible set of low-level, orthogonal parameters capable of combining with the ICL goal language to fully express how information should flow among participants. In certain preferred embodiments of the present invention, multiple parameters can be grouped together and given a group name. The resulting high-level advice parameters can preferably be used to express concepts analogous to KQML's performatives, as well as define classifications of problem types. For instance, KQML's "ask_all" and "ask_one" performatives would be represented as combinations of values given to the parameters reply, parallel ok, and solution_limit. As an example of a higher-level problem type, the strategy "math_problem" might preferably send the query to all appropriate math solvers in parallel, collect their responses, and signal a conflict if different answers are returned. The strategy "essay_question" might preferably send the request to all appropriate participants, and signal a problem (i.e., cheating) if any of the returned answers are identical.

Facilitation

In a preferred embodiment of the present invention, when a facilitator receives a compound goal, its job is to construct a goal satisfaction plan and oversee its satisfaction in an optimal or near optimal manner that is consistent with the specified advice. The facilitator of the present invention maintains a knowledge base that records the capabilities of a collection of agents, and uses that knowledge to assist requesters and providers of services in making contact.

FIG. 7 schematically shows data structures 700 internal to a facilitator in accordance with one embodiment of the present invention. Consider the function of an Agent Registry 702 in the present invention. Each registered agent may be seen as associated with a collection of fields found within its parent facilitator such as shown in the figure. Each registered agent may optionally possess a Symbolic Name which would be entered into field 704. As mentioned elsewhere, Symbolic Names need not be unique to each instance of an agent. Note that an agent may in certain preferred embodiments of the present invention possess more than one Symbolic Name. Such Symbolic Names would each be found through their associations in the Agent Registry entries. Each agent, when registered, must possess a Unique Address, which is entered into the Unique Address field 706.

With further reference to FIG. 7, each registered agent may be optionally associated with one or more capabilities, which have associated Capability Declaration fields 708 in the parent facilitator Agent Registry 702. These capabilities may define not just functionality, but may further provide a utility parameter indicating, in some manner (e.g., speed,

US 6,851,115 B1

17

accuracy, etc), how effective the agent is at providing the declared capability. Each registered agent may be optionally associated with one or more data components, which have associated Data Declaration fields **710** in the parent facilitator Agent Registry **702**. Each registered agent may be optionally associated with one or more triggers, which preferably could be referenced through their associated Trigger Declaration fields **712** in the parent facilitator Agent Registry **702**. Each registered agent may be optionally associated with one or more tasks, which preferably could be referenced through their associated Task Declaration fields **714** in the parent facilitator Agent Registry **702**. Each registered agent may be optionally associated with one or more Process Characteristics, which preferably could be referenced through their associated Process Characteristics Declaration fields **716** in the parent facilitator Agent Registry **702**. Note that these characteristics in certain preferred embodiments of the present invention may include one or more of the following: Machine Type (specifying what type of computer may run the agent), Language (both computer and human interface).

A facilitator agent in certain preferred embodiments of the present invention further includes a Global Persistent Database **720**. The database **720** is composed of data elements which do not rely upon the invocation or instantiation of client agents for those data elements to persist. Examples of data elements which might be present in such a database include but are not limited to the network address of the facilitator agent's server, facilitator agent's server accessible network port list, firewalls, user lists, and security options regarding the access of server resources accessible to the facilitator agent.

A simplified walk through of operations involved in creating a client agent, a client agent initiating a service request, a client agent responding to a service request and a facilitator agent responding to a service request are including hereafter by way of illustrating the use of such a system. These figures and their accompanying discussion are provided by way of illustration of one preferred embodiment of the present invention and are not intended to limit the scope of the present invention.

FIG. **8** depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention. The operations begin with starting the Agent Registration in a step **800**. In a next step **802**, the Installer, such as a client or facilitator agent, invokes a new client agent. It will be appreciated that any computer entity is capable of invoking a new agent. The system then instantiates the new client agent in a step **804**. This operation may involve resource allocations somewhere in the network on a local computer system for the client agent, which will often include memory as well as placement of references to the newly instantiated client agent in internal system lists of agents within that local computing system. Once instantiated, the new client and its parent facilitator establish a communications link in a step **806**. In certain preferred embodiments, this communications link involves selection of one or more physical transport mechanisms for this communication. Once established, the client agent transmits its profile to the parent facilitator in a step **808**. When received, the parent facilitator registers the client agent in a step **810**. Then, at a step **812**, a client agent has been instantiated in accordance with one preferred embodiment of the present invention.

FIG. **9** depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a preferred embodiment

18

of the present invention. The method of FIG. **9** begins in a step **900**, wherein any initialization or other such procedures may be performed. Then, in a step **902**, the client agent determines a goal to be achieved (or solved). This goal is then translated in a step **904** into ICL, if it is not already formulated in it. The goal, now stated in ICL, is then transmitted to the client agent's parent facilitator in a step **906**. The parent facilitator responds to this service request and at a later time, the client agent receives the results of the request in a step **908**, operations of FIG. **9** being complete in a done step **910**.

FIG. **10** depicts operations involved in a client agent responding to a service request in accordance with a preferred embodiment of the present invention. Once started in a step **1000**, the client agent receives the service request in a step **1002**. In a next step **1004**, the client agent parses the received request from ICL. The client agent then determines if the service is available in a step **1006**. If it is not, the client agent returns a status report to that effect in a step **1008**. If the service is available, control is passed to a step **1010** where the client performs the requested service. Note that in completing step **1010** the client may form complex goal expressions, requesting results for these solvables from the facilitator agent. For example, a fax agent might fax a document to a certain person only after requesting and receiving a fax number for that person. Subsequently, the client agent either returns the results of the service and/or a status report in a step **1012**. The operations of FIG. **10** are complete in a done step **1014**.

FIG. **11** depicts operations involved in a facilitator agent response to a service request in accordance with a preferred embodiment of the present invention. The start of such operations in step **1100** leads to the reception of a goal request in a step **1102** by the facilitator. This request is then parsed and interpreted by the facilitator in a step **1104**. The facilitator then proceeds to construct a goal satisfaction plan in a next step **1106**. In steps **1108** and **1110**, respectively, the facilitator determines the required sub-goals and then selects agents suitable for performing the required sub-goals. The facilitator then transmits the sub-goal requests to the selected agents in a step **1112** and receives the results of these transmitted requests in a step **1114**. It should be noted that the actual implementation of steps **1112** and **1114** are dependent upon the specific goal satisfaction plan. For instance, certain sub-goals may be sent to separate agents in parallel, while transmission of other sub-goals may be postponed until receipt of particular answers. Further, certain requests may generate multiple responses that generate additional sub-goals. Once the responses have been received, the facilitator determines whether the original requested goal has been completed in a step **1118**. If the original requested goal has not been completed, the facilitator recursively repeats the operations **1106** through **1116**. Once the original requested goal is completed, the facilitator returns the results to the requesting agent **1118** and the operations are done at **1120**.

A further preferred embodiment of the present invention incorporates transparent delegation, which means that a requesting agent can generate a request, and a facilitator can manage the satisfaction of that request, without the requester needing to have any knowledge of the identities or locations of the satisfying agents. In some cases, such as when the request is a data query, the requesting agent may also be oblivious to the number of agents involved in satisfying a request. Transparent delegation is possible because agents' capabilities (solvables) are treated as an abstract description of a service, rather than as an entry point into a library or body of code.

US 6,851,115 B1

19

A further preferred embodiment of the present invention incorporates facilitator handling of compound goals, preferably involving three types of processing: delegation, optimization and interpretation.

Delegation processing preferably supports facilitator determination of which specific agents will execute a compound goal and how such a compound goal's sub-goals will be combined and the sub-goal results routed. Delegation involves selective application of global and local constraint and advice parameters onto the specific sub-goals. Delegation results in a goal that is unambiguous as to its meaning and as to the agents that will participate in satisfying it.

Optimization processing of the completed goal preferably includes the facilitator using sub-goal parallelization where appropriate. Optimization results in a goal whose interpretation will require as few exchanges as possible, between the facilitator and the satisfying agents, and can exploit parallel efforts of the satisfying agents, wherever this does not affect the goal's meaning.

Interpretation processing of the optimized goal. Completing the addressing of a goal involves the selection of one or more agents to handle each of its sub-goals (that is, each sub-goal for which this selection has not been specified by the requester). In doing this, the facilitator uses its knowledge of the capabilities of its client agents (and possibly of other facilitators, in a multi-facilitator system). It may also use strategies or advice specified by the requester, as explained below. The interpretation of a goal involves the coordination of requests to the satisfying agents, and assembling their responses into a coherent whole, for return to the requester.

A further preferred embodiment of present invention extends facilitation so the facilitator can employ strategies and advice given by the requesting agent, resulting in a variety of interaction patterns that may be instantiated in the satisfaction of a request.

A further preferred embodiment of present invention handles the distribution of both data update requests and requests for installation of triggers, preferably using some of the same strategies that are employed in the delegation of service requests.

Note that the reliance on facilitation is not absolute; that is, there is no hard requirement that requests and services be matched up by the facilitator, or that interagent communications go through the facilitator. There is preferably support in the agent library for explicit addressing of requests. However, a preferred embodiment of the present invention encourages employment of the paradigm of agent communities, minimizing their development effort, by taking advantage of the facilitator's provision of transparent delegation and handling of compound goals.

A facilitator is preferably viewed as a coordinator, not a controller, of cooperative task completion. A facilitator preferably never initiates an activity. A facilitator preferably responds to requests to manage the satisfaction of some goal, the update of some data repository, or the installation of a trigger by the appropriate agent or agents. All agents can preferably take advantage of the facilitator's expertise in delegation, and its up-to-date knowledge about the current membership of a dynamic community. The facilitator's coordination services often allows the developer to lessen the complexity of individual agents, resulting in a more manageable software development process, and enabling the creation of lightweight agents.

Maintaining Data Repositories

The agent library supports the creation, maintenance, and use of databases, in the form of data solvables. Creation of

20

a data solvable requires only that it be declared. Querying a data solvable, as with access to any solvable, is done using `oaa_Solve`.

A data solvable is conceptually similar to a relation in a relational database. The facts associated with each solvable are maintained by the agent library, which also handles incoming messages containing queries of data solvables. The default behavior of an agent library in managing these facts may preferably be refined, using parameters specified with the solvable's declaration. For example, the parameter `single_value` preferably indicates that the solvable should only contain a single fact at any given point in time. The parameter `unique_values` preferably indicates that no duplicate values should be stored.

Other parameters preferably allow data solvables use of the concepts of ownership and persistence. For implementing shared repositories, it is often preferable to maintain a record of which agent created each fact of a data solvable with the creating agent being preferably considered the fact's owner. In many applications, it is preferable to remove an agent's facts when that agent goes offline (for instance, when the agent is no longer participating in the agent community, whether by deliberate termination or by malfunction). When a data solvable is declared to be non-persistent, its facts are automatically maintained in this way, whereas a persistent data solvable preferably retains its facts until they are explicitly removed.

A further preferred embodiment of present invention supports an agent library through procedures by which agents can update (add, remove, and replace) facts belonging to data solvables, either locally or on other agents, given that they have preferably the required permissions. These procedures may preferably be refined using many of the same parameters that apply to service requests. For example, the address parameter preferably specifies one or more particular agents to which the update request applies. In its absence, just as with service requests, the update request preferably goes to all agents providing the relevant data solvable. This default behavior can be used to maintain coordinated "mirror" copies of a data set within multiple agents, and can be useful in support of distributed, collaborative activities.

Similarly, the feedback parameters, described in connection with `oaa_Solve`, are preferably available for use with data maintenance requests.

A further preferred embodiment of present invention supports ability to provide data solvables not just to client agents, but also to facilitator agents. Data solvables can preferably be created, maintained and used by a facilitator. The facilitator preferably can, at the request of a client of the facilitator, create, maintain and share the use of data solvables with all the facilitator's clients. This can be useful with relatively stable collections of agents, where the facilitator's workload is predictable.

Using a Blackboard Style of Communication

In a further preferred embodiment of present invention, when a data solvable is publicly readable and writable, it acts essentially as a global data repository and can be used cooperatively by a group of agents. In combination with the use of triggers, this allows the agents to organize their efforts around a "blackboard" style of communication.

As an example, the "DCG-NL" agent (one of several existing natural language processing agents), provides natural language processing services for a variety of its peer agents, expects those other agents to record, on the facilitator, the vocabulary to which they are prepared to respond, with an indication of each word's part of speech,

US 6,851,115 B1

21

and of the logical form (ICL sub-goal) that should result from the use of that word. In a further preferred embodiment of present invention, the NL agent, preferably when it comes online, preferably installs a data solvable for each basic part of speech on its facilitator. For instance, one such solvable would be:

solvable(noun(Meaning, Syntax), [], [])

Note that the empty lists for the solvable's permissions and parameters are acceptable here, since the default permissions and parameters provide appropriate functionality.

A further preferred embodiment of present invention incorporating an Office Assistant system as discussed herein or similar to the discussion here supports several agents making use of these or similar services. For instance, the database agent uses the following call, to library procedure oaa_AddData, to post the noun 'boss', and to indicate that the "meaning" of boss is the concept 'manager':

oaa_AddData(noun(manager, atom(boss)), [address (parent)])

Autonomous Monitoring with Triggers

A further preferred embodiment of present invention includes support for triggers, providing a general mechanism for requesting some action be taken when a set of conditions is met. Each agent can preferably install triggers either locally, for itself, or remotely, on its facilitator or peer agents. There are preferably at least four types of triggers: communication, data, task, and time. In addition to a type, each trigger preferably specifies at least a condition and an action, both preferably expressed in ICL. The condition indicates under what circumstances the trigger should fire, and the action indicates what should happen when it fires. In addition, each trigger can be set to fire either an unlimited number of times, or a specified number of times, which can be any positive integer.

Triggers can be used in a variety of ways within preferred embodiments of the present invention. For example, triggers can be used for monitoring external sensors in the execution environment, tracking the progress of complex tasks, or coordinating communications between agents that are essential for the synchronization of related tasks. The installation of a trigger within an agent can be thought of as a representation of that agent's commitment to carry out the specified action, whenever the specified condition holds true.

Communication triggers preferably allow any incoming or outgoing event (message) to be monitored. For instance, a simple communication trigger may say something like: "Whenever a solution to a goal is returned from the facilitator, send the result to the presentation manager to be displayed to the user."

Data triggers preferably monitor the state of a data repository (which can be maintained on a facilitator or a client agent). Data triggers' conditions may be tested upon the addition, removal, or replacement of a fact belonging to a data solvable. An example data trigger is: "When 15 users are simultaneously logged on to a machine, send an alert message to the system administrator."

Task triggers preferably contain conditions that are tested after the processing of each incoming event and whenever a timeout occurs in the event polling. These conditions may specify any goal executable by the local ICL interpreter, and most often are used to test when some solvable becomes satisfiable. Task triggers are useful in checking for task-specific internal conditions. Although many cases such conditions are captured by solvables, in other cases they may not be. For example, a mail agent might watch for new incoming mail, or an airline database agent may monitor

22

which flights will arrive later than scheduled. An example task trigger is: "When mail arrives for me about security, notify me immediately."

Time triggers preferably monitor time conditions. For instance, an alarm trigger can be set to fire at a single fixed point in time (e.g., "On December 23rd at 3 pm"), or on a recurring basis (e.g., "Every three minutes from now until noon").

Triggers are preferably implemented as data solvables, declared implicitly for every agent. When requesting that a trigger be installed, an agent may use many of the same parameters that apply to service and data maintenance requests.

A further preferred embodiment of present invention incorporates semantic support, in contrast with most programming methodologies, of the agent on which the trigger is installed only having to know how to evaluate the conditional part of the trigger, not the consequence. When the trigger fires, the action is delegated to the facilitator for execution. Whereas many commercial mail programs allow rules of the form "When mail arrives about XXX, [forward it, delete it, archive it]", the possible actions are hard-coded and the user must select from a fixed set.

A further preferred embodiment of present invention, the consequence of a trigger may be any compound goal executable by the dynamic community of agents. Since new agents preferably define both functionality and vocabulary, when an unanticipated agent (for example, a fax agent) joins the community, no modifications to existing code is required for a user to make use of it—"When mail arrives, fax it to Bill Smith."

The Agent Library

In a preferred embodiment of present invention, the agent library provides the infrastructure for constructing an agent-based system. The essential elements of protocol (involving the details of the messages that encapsulate a service request and its response) are preferably made transparent to simplify the programming applications. This enables the developer to focus functionality, rather than message construction details and communication details. For example, to request a service of another agent, an agent preferably calls the library procedure oaa_Solve. This call results in a message to a facilitator, which will exchange messages with one or more service providers, and then send a message containing the desired results to the requesting agent. These results are returned via one of the arguments of oaa_Solve. None of the messages involved in this scenario is explicitly constructed by the agent developer. Note that this describes the synchronous use of oaa_Solve.

In another preferred embodiment of present invention, an agent library provides both intraagent and interagent infrastructure; that is, mechanisms supporting the internal structure of individual agents, on the one hand, and mechanisms of cooperative interoperation between agents, on the other. Note that most of the infrastructure cuts across this boundary with many of the same mechanisms supporting both agent internals and agent interactions in an integrated fashion. For example, services provided by an agent preferably can be accessed by that agent through the same procedure (oaa_Solve) that it would employ to request a service of another agent (the only difference being in the address parameter accompanying the request). This helps the developer to reuse code and avoid redundant entry points into the same functionality.

Both of the preferred characteristics described above (transparent construction of messages and integration of intraagent with interagent mechanisms) apply to most other

US 6,851,115 B1

23

library functionality as well, including but not limited to data management and temporal control mechanisms.

Source Code Appendix

Source code for version 2.0 of the OAA software product is included as an appendix hereto, and is incorporated herein by reference. The code includes an agent library, which provides infrastructure for constructing an agent-based system. The library's several families of procedures provide the functionalities discussed above, as well as others that have not been discussed here but that will be sufficiently clear to the interested practitioner. For example, declarations of an agent's solvables, and their registration with a facilitator, are managed using procedures such as `oaa_Declare`, `oaa_Undeclare`, and `oaa_Redeclare`. Updates to data solvables can be accomplished with a family of procedures including `oaa_AddData`, `oaa_RemoveData`, and `oaa_ReplaceData`. Similarly, triggers are maintained using procedures such as `oaa_AddTrigger`, `oaa_RemoveTrigger`, and `oaa_ReplaceTrigger`. The provided source code also includes source code for an OAA Facilitator Agent.

The source code appendix is offered solely as a means of further helping practitioners to construct a preferred embodiment of the invention. By no means is the source code intended to limit the scope of the present invention.

Illustrative Applications

To further illustrate the technology of the preferred embodiment, we will next present and discuss two sample applications of the present inventions.

Unified Messaging

A further preferred embodiment of present invention incorporates a Unified Messaging application extending the Automated Office application presented previously herein with an emphasis on ubiquitous access and dynamic presentation of the information and services supported by the agent community. The agents used in this application are depicted in FIG. 12.

A hypothetical example of realistic dialog using a preferred embodiment of the present invention can provide insight into how systems may preferably be built using the present invention. In this scenario, the user, with only a telephone as an interface, is planning a trip to Boston where he will soon give a presentation. Capitalized sentences are phrases spoken by the user into the telephone and processed by a phone agent 452.

Responses, unless otherwise indicated, are spoken by the system using text-to-speech generation agent 454.

1.1 Welcome to SRI International. Please enter your user ID and password.

<User enters touchtone ID and password>

Good to see you again Adam Cheyer. I am listening to you.

Every user interface agent 408, including the telephone agent 452, should know the identify of its user. This information is used in resolving anaphoric references such as "Me" and "I", and allows multiple user interfaces operated by the same user to work together.

1.2 WHAT IS TODAY'S SCHEDULE?

Here is today's schedule for Adam Cheyer:

At 10 am for 1 hour, meeting with Dave.

At 3 pm for 1 hour, presentation about software agents.

End of schedule.

If the user is operating both a graphical user interface and a telephone, as described in conjunction with the Automated Office application, the result of this spoken request is to display a calendar window containing the current schedule. In this case, with no graphical display available, the GEN_NL agent 1202 is tasked to produce a spoken response that can be played over the phone. GEN_NL shares the same dynamic vocabulary and phrasal rules as the natural language parser DCG_NL 426, and contains strategies for

24

producing responses to queries using either simple or list-based multimedia utterances.

1.3 FIND FRIDAY'S WEATHER IN BOSTON.

The weather in Boston for Friday is as follows:

Sunny in the morning. Partly cloudy in the afternoon with a 20 percent chance of thunderstorms late. Highs in the mid 70s.

In addition to data accessible from legacy applications, content may be retrieved by web-reading agents which provide wrappers around useful websites.

1.4 FIND ALL NEW MAIL MESSAGES.

There are 2 messages available.

Message 1, from Mark Tierny, entitled "OAA meeting."

1.5 NEXT MESSAGE

Message 2, from Jennifer Schweffler, entitled "Presentation Summary."

1.6 PLAY IT.

This message is a multipart MIME-encoded message.

There are two parts.

Part 1. (Voicemail message, not text-to speech):

Thanks for taking part as a speaker in our conference.

The schedule will be posted soon on our homepage.

1.7 NEXT PART

Part 2. (read using text-to-speech):

The presentation home page is <http://www...>

1.8 PRINT MESSAGE

Command executed.

Mail messages are no longer just simple text documents, but often consist of multiple subparts containing audio files, pictures, webpages, attachments and so forth. When a user asks to play a complex email message over the telephone, many different agents may be implicated in the translation process, which would be quite different given the request "print it." The challenge is to develop a system which will enable agents to cooperate in an extensible, flexible manner that alleviates explicit coding of agent interactions for every possible input/output combination.

In a preferred embodiment of the present invention, each agent concentrates only on what it can do and on what it knows, and leaves other work to be delegated to the agent community. For instance, a printer agent 1204, defining the solvable `print(Object,Parameters)`, can be defined by the following pseudo-code, which basically says, "If someone can get me a document, in either POSTSCRIPT or text form, I can print it."

```
print(Object, Parameters) {
  ' If Object is reference to "it", find an appropriate document
  if (Object == "ref(it)")
    oaa_Solve(resolve_reference(the, document, Params, Object), [ ]);
  ' Given a reference to some document, ask for the document in POSTSCRIPT
  if (Object == "id(Pointer)")
    oaa_Solve(resolve_id_as(id(Pointer), postscript, [ ], Object), [ ]);
  ' If Object is of type text or POSTSCRIPT, we can print it.
  if ((Object is of type Text) or (Object is of type Postscript))
    do print (Object);
}
```

In the above example, since an email message is the salient document, the mail agent 442 will receive a request

US 6,851,115 B1

25

to produce the message as POSTSCRIPT. Whereas the mail agent 442 may know how to save a text message as POSTSCRIPT, it will not know what to do with a webpage or voicemail message. For these parts of the message, it will simply send oaa_Solve requests to see if another agent knows how to accomplish the task.

Until now, the user has been using only a telephone as user interface. Now, he moves to his desktop, starts a web browser 436, and accesses the URL referenced by the mail message.

1.9 RECORD MESSAGE

Recording voice message. Start speaking now.

1.10 THIS IS THE UPDATED WEB PAGE CONTAINING THE PRESENTATION SCHEDULE.

Message one recorded.

1.11 IF THIS WEB PAGE CHANGES, GET IT TO ME WITH NOTE ONE.

Trigger added as requested.

In this example, a local agent 436 which interfaces with the web browser can return the current page as a solution to the request "oaa_Solve(resolve_reference(this, web_page, [], Ref), [])", sent by the NL agent 426. A trigger is installed on a web agent 436 to monitor changes to the page, and when the page is updated, the notify agent 446 can find the user and transmit the webpage and voicemail message using the most appropriate media transfer mechanism.

This example based on the Unified Messaging application is intended to show how concepts in accordance with the present invention can be used to produce a simple yet extensible solution to a multi-agent problem that would be difficult to implement using a more rigid framework. The application supports adaptable presentation for queries across dynamically changing, complex information; shared context and reference resolution among applications; and flexible translation of multimedia data. In the next section, we will present an application which highlights the use of parallel competition and cooperation among agents during multi-modal fusion.

Multimodal Map

A further preferred embodiment of present invention incorporates the Multimodal Map application. This application demonstrates natural ways of communicating with a community of agents, providing an interactive interface on which the user may draw, write or speak. In a travel-planning domain illustrated by FIG. 13, available information includes hotel, restaurant, and tourist-site data retrieved by distributed software agents from commercial Internet sites. Some preferred types of user interactions and multimodal issues handled by the application are illustrated by a brief scenario featuring working examples taken from the current system.

Sara is planning a business trip to San Francisco, but would like to schedule some activities for the weekend while she is there. She turns on her laptop PC, executes a map application, and selects San Francisco.

2.1 [Speaking] Where is downtown?

Map scrolls to appropriate area.

2.2 [Speaking and drawing region] Show me all hotels near here.

Icons representing hotels appear.

2.3 [Writes on a hotel] Info?

A textual description (price, attributes, etc.) appears.

2.4 [Speaking] I only want hotels with a pool. Some hotels disappear.

2.5 [Draws a crosscut on a hotel that is too close to a highway)

26

Hotel disappears

2.6 [Speaking and circling] Show me a photo of this hotel.

Photo appears.

2.7 (Points to another hotel)

Photo appears.

2.8 [Speaking] Price of the other hotel?

Price appears for previous hotel.

2.9 [Speaking and drawing an arrow] Scroll down.

Display adjusted.

2.10 [Speaking and drawing an arrow toward a hotel]

What is the distance from this hotel to Fisherman's Wharf?

Distance displayed.

15 2.11 [Pointing to another place and speaking] And the distance to here?

Distance displayed.

Sara decides she could use some human advice. She picks up the phone, calls Bob, her travel agent, and writes Start collaboration to synchronize his display with hers. At this point, both are presented with identical maps, and the input and actions of one will be remotely seen by the other.

3.1 [Sara speaks and circles two hotels]

Bob, I'm trying to choose between these two hotels.

25 Any opinions?

3.2 [Bob draws an arrow, speaks, and points]

Well, this area is really nice to visit. You can walk there from this hotel.

30 Map scrolls to indicated area. Hotel selected.

3.3 [Sara speaks] Do you think I should visit Alcatraz?

3.4 [Bob speaks] Map, show video of Alcatraz.

Video appears.

35 3.5 [Bob speaks] Yes, Alcatraz is a lot of fun.

A further preferred embodiment of present invention generates the most appropriate interpretation for the incoming streams of multimodal input. Besides providing a user interface to a dynamic set of distributed agents, the application is preferably built using an agent framework. The present invention also contemplates aiding the coordinate competition and cooperation among information sources, which in turn works in parallel to resolve the ambiguities arising at every level of the interpretation process: low-level processing of the data stream, anaphora resolution, cross-modality influences and addressee.

Low-level processing of the data stream: Pen input may be preferably interpreted as a gesture (e.g., 2.5: cross-out) by one algorithm, or as handwriting by a separate recognition process (e.g., 2.3: "info?"). Multiple hypotheses may preferably be returned by a modality recognition component.

Anaphora resolution: When resolving anaphoric references, separate information sources may contribute to resolving the reference: context by object type, deictic, visual context, database queries, discourse analysis. An example of information provided through context by object type is found in interpreting an utterance such as "show photo of the hotel", where the natural language component can return a list of the last hotels talked about. Deictic information in combination with a spoken utterance like "show photo of this hotel" may preferably include pointing, circling, or arrow gestures which might indicate the desired object (e.g., 2.7). Deictic references may preferably occur before, during, or after an accompanying verbal command.

65 Information provided in a visual context, given for the request "display photo of the hotel" may preferably include the user interface agent might determine that only one hotel

US 6,851,115 B1

27

is currently visible on the map, and therefore this might be the desired reference object. Database queries preferably involving information from a database agent combined with results from other resolution strategies. Examples are "show me a photo of the hotel in Menlo Park" and 2.2. Discourse analysis preferably provides a source of information for phrases such as "No, the other one" (or 2.8).

The above list of preferred anaphora resolution mechanisms is not exhaustive. Examples of other preferred resolution methods include but are not limited to spatial reasoning ("the hotel between Fisherman's Wharf and Lombard Street") and user preferences ("near my favorite restaurant").

Cross-modality influences: When multiple modalities are used together, one modality may preferably reinforce or remove or diminish ambiguity from the interpretation of another. For instance, the interpretation of an arrow gesture may vary when accompanied by different verbal commands (e.g., "scroll left" vs. "show info about this hotel"). In the latter example, the system must take into account how accurately and unambiguously an arrow selects a single hotel.

Addressee: With the addition of collaboration technology, humans and automated agents all share the same workspace. A pen doodle or a spoken utterance may be meant for either another human, the system (3.1), or both (3.2).

The implementation of the Multimodal Map application illustrates and exploits several preferred features of the present invention: reference resolution and task delegation by parallel parameters of oaa_Solve, basic multi-user collaboration handled through built-in data management services, additional functionality readily achieved by adding new agents to the community, domain-specific code cleanly separated from other agents.

A further preferred embodiment of present invention provides reference resolution and task delegation handled in a distributed fashion by the parallel parameters of oaa_Solve, with meta-agents encoding rules to help the facilitator make context- or user-specific decisions about priorities among knowledge sources.

A further preferred embodiment of present invention provides basic multi-user collaboration handled through at least one built-in data management service. The map user interface preferably publishes data solvables for elements such as icons, screen position, and viewers, and preferably defines these elements to have the attribute "shareable". For every update to this public data, the changes are preferably automatically replicated to all members of the collaborative session, with associated callbacks producing the visible effect of the data change (e.g., adding or removing an icon).

Functionality for recording and playback of a session is preferably implemented by adding agents as members of the collaborative community. These agents either record the data changes to disk, or read a log file and replicate the changes in the shared environment.

The domain-specific code for interpreting travel planning dialog is preferably separated from the speech, natural language, pen recognition, database and map user interface agents. These components were preferably reused without modification to add multimodal map capabilities to other applications for activities such as crisis management, multi-robot control, and the MVEWS tools for the video analyst. Improved Scalability and Fault Tolerance

Implementations of a preferred embodiment of present invention which rely upon simple, single facilitator architectures may face certain limitations with respect to scalability, because the single facilitator may become a

28

communications bottleneck and may also represent a single, critical point for system failure.

Multiple facilitator systems as disclosed in the preferred embodiments to this point can be used to construct peer-to-peer agent networks as illustrated in FIG. 14. While such embodiments are scalable, they do possess the potential for communication bottlenecks as discussed in the previous paragraph and they further possess the potential for reliability problems as central, critical points of vulnerability to systems failure.

A further embodiment of present invention supports a facilitator implemented as an agent like any other, whereby multiple facilitator network topologies can be readily constructed. One example configuration (but not the only possibility) is a hierarchical topology as depicted in FIG. 15, where a top level Facilitator manages collections of both client agents 1508 and other Facilitators, 1504 and 1506. Facilitator agents could be installed for individual users, for a group of users, or as appropriate for the task.

Note further, that network work topologies of facilitators can be seen as graphs where each node corresponds to an instance of a facilitator and each edge connecting two or more nodes corresponds to a transmission path across one or more physical transport mechanisms. Some nodes may represent facilitators and some nodes may represent clients. Each node can be further annotated with attributes corresponding to include triggers, data, capabilities but not limited to these attributes.

A further embodiment of present invention provides enhanced scalability and robustness by separating the planning and execution components of the facilitator. In contrast with the centralized facilitation schemes described above, the facilitator system 1600 of FIG. 16 separates the registry/planning component from the execution component. As a result, no single facilitator agent must carry all communications nor does the failure of a single facilitator agent shut down the entire system.

Turning directly to FIG. 16, the facilitator system 1600 includes a registry/planner 1602 and a plurality of client agents 1612-1616. The registry/planner 1604 is typically replicated in one or more locations accessible by the client agents. Thus if the registry/planner 1604 becomes unavailable, the client agents can access the replicated registry/planner(s).

This system operates, for example, as follows. An agent transmits a goal 1610 to the registry planner 1602. The registry/planner 1604 translates the goal into an unambiguous execution plan detailing how to accomplish any sub-goals developed from the compound goal, as well as specifying the agents selected for performing the sub-goals. This execution plan is provided to the requesting agent which in turn initiates peer-to-peer interactions 1618 in order to implement the detailed execution plan, routing and combining information as specified within the execution plan. Communication is distributed thus decreasing sensitivity of the system to bandwidth limitations of a single facilitator agent. Execution state is likewise distributed thus enabling system operation even when a facilitator agent fails.

Further embodiments of present invention incorporate into the facilitator functionality such as load-balancing, resource management, and dynamic configuration of agent locations and numbers, using (for example) any of the topologies discussed. Other embodiments incorporate into a facilitator the ability to aid agents in establishing peer-to-peer communications. That is, for tasks requiring a sequence of exchanges between two agents, the facilitator assists the agents in finding one another and establishing

US 6,851,115 B1

29

communication, stepping out of the way while the agents communicate peer-to-peer over a direct, perhaps dedicated channel.

Further preferred embodiments of the present invention incorporate mechanisms for basic transaction management, such as periodically saving the state of agents (both facilitator and client) and rolling back to the latest saved state in the event of the failure of an agent.

What is claimed is:

1. A computer-implemented method for communication and cooperative task completion among a plurality of distributed electronic agents, comprising the acts of:

registering a description of each active client agent's functional capabilities as corresponding registered functional capabilities, using an expandable, platform-independent, inter-agent language, wherein the inter-agent language includes:

a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events;

a content layer comprising one or more of goals, triggers and data elements associated with the events;

receiving a request for service as a base goal in the inter-agent language, in the form of an arbitrarily complex goal expression; and

dynamically interpreting the arbitrarily complex goal expression, said act of interpreting further comprising: generating one or more sub-goals expressed in the inter-agent language;

constructing a goal satisfaction plan wherein the goal satisfaction plan includes:

a suitable delegation of sub-goal requests to best complete the requested service request—by using reasoning that includes one or more of domain-independent coordination strategies, domain-specific reasoning, and application-specific reasoning comprising rules and learning algorithms; and

dispatching each of the sub-goals to a selected client agent for performance, based on a match between the sub-goal being dispatched and the registered functional capabilities of the selected client agent.

2. A computer-implemented method as recited in claim 1, further including the following acts of:

receiving a new request for service as a base goal using the inter-agent language, in the form of another arbitrarily complex goal expression, from at least one of the selected client agents in response to the sub-goal dispatched to said agent; and

recursively applying the step of dynamically interpreting the arbitrarily complex goal expression in order to perform the new request for service.

3. A computer-implemented method as recited in claim 2 wherein the act of registering a specific agent further includes:

invoking the specific agent in order to activate the specific agent;

instantiating an instance of the specific agent; and

transmitting the new agent profile from the specific agent to a facilitator agent in response to the instantiation of the specific agent.

4. A computer-implemented method as recited in claim 1 further including the act of deactivating a specific client agent no longer available to provide services by deleting the registration of the specific client agent.

30

5. A computer-implemented method as recited in claim 1 further comprising the act of providing an agent registry data structure.

6. A computer-implemented method as recited in claim 5 wherein the agent registry data structure includes at least one symbolic name for each active agent.

7. A computer-implemented method of recited in claim 5 wherein the agent registry data structure includes at least one data declaration for each active agent.

8. A computer-implemented method as recited in claim 5 wherein the agent registry data structure includes at least one trigger declaration for one active agent.

9. A computer-implemented method as recited in claim 5 wherein the agent registry data structure includes at least one task declaration, and process characteristics for each active agent.

10. A computer-implemented method as recited in claim 5 wherein the agent registry data structure includes at least one process characteristic for each active agent.

11. A computer-implemented method as recited in claim 1 further comprising the act of establishing communication between the plurality of distributed agents.

12. A computer-implemented method as recited in claim 1 further comprising the acts of:

receiving a request for service in a second language differing from the inter-agent language;

selecting a registered agent capable of converting the second language into the inter-agent language; and

forwarding the request for service in a second language to the registered agent capable of converting the second language into the inter-agent language, implicitly requesting that such a conversion be performed and the results returned.

13. A computer-implemented method as recited in claim 12 wherein the request include a natural language query, and the second registered agent capable of converting the second language into the inter-agent language service is a natural language agent.

14. A computer-implemented method as recited in claim 13 wherein the natural language query was generated by a user interface agent.

15. A computer-implemented method as recited in claim 1, wherein the base goal requires setting a trigger having conditional functionality and consequential functionality.

16. A computer-implemented method as recited in claim 15 wherein the trigger is an outgoing communications trigger, the computer implemented method further including the acts of:

monitoring all outgoing communication events in order to determine whether a specific outgoing communication event has occurred; and

in response to the occurrence of the specific outgoing communication event, performing the particular action defined by the trigger.

17. A computer-implemented method as recited in claim 15 wherein the trigger is an incoming communications trigger, the computer implemented method further including the acts of:

monitoring all incoming communication events in order to determine whether a specific incoming communication event has occurred; and

in response to the occurrence of a specific incoming communication event satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

US 6,851,115 B1

31

18. A computer-implemented method as recited in claim 15 wherein the trigger is a data trigger, the computer implemented method further including the acts of:

monitoring a state of a data repository; and
in response to a particular state event satisfying the trigger conditional functionality, performing the particular consequential functionality defined by the trigger.

19. A computer-implemented method as recited in claim 15 wherein the trigger is a time trigger, the computer implemented method further including the acts of:

monitoring for the occurrence of a particular time condition; and
in response to the occurrence of a particular time condition satisfying the trigger conditional functionality performing the particular consequential functionality defined by the trigger.

20. A computer-implemented method as recited in claim 15 wherein the trigger is installed and executed within the facilitator agent.

21. A computer-implemented method as recited in claim 15 wherein the trigger is installed and executed, within a first service-providing agent.

22. A computer-implemented method as recited in claim 15 wherein the conditional functionality of the trigger is installed on a facilitator agent.

23. A computer-implemented method as recited in claim 22 wherein the consequential functionality is installed on a specific service-providing agent other than a facilitator agent.

24. A computer-implemented method as recited in claim 15 wherein the conditional functionality of the trigger is installed on specific service-providing agent other than a facilitator agent.

25. A computer-implemented method as recited in claim 15 wherein the consequential functionality of the trigger is installed on a facilitator agent.

26. A computer-method as recited in claim 1 wherein the base goal is a compound goal having sub-goals separated by operators.

27. A computer-implemented method as recited in claim 26 wherein the type of available operators includes a conjunction operator, a disjunction operator, and a conditional execution operator.

28. A computer-implemented method as recited in claim 27 wherein the type of available operators further includes a parallel disjunction operator that indicates that disjunct goals are to be performed by different agents.

29. A computer program stored on a computer readable medium, the computer program executable to facilitate cooperative task completion within a distributed computing environment, the distributed computing environment including a plurality of autonomous electronic agents, the distributed computing environment supporting an Interagent Communication Language, the computer program comprising computer executable instructions for:

providing an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment;

interpreting a service request in order to determine a base goal that may be a compound, arbitrarily complex base goal, the service request adhering to an Interagent Communication Language (ICL), where in the ICL includes:

a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events; and

32

a content layer comprising one or more of goals, triggers and data elements associated with the events;

the act of interpreting including the sub-acts of:

determining any task completion advice provided by the base goal, and
determining any task completion constraints provided by the base goal;

constructing a base goal satisfaction plan including the sub-acts of:

determining whether the request service is available,
determining sub-goals required in completing the base goal by using reasoning that includes one or more of domain-independent coordination strategies, domain-specific reasoning, and application-specific reasoning comprising rules and learning algorithms, selecting service-providing electronic agents from the agent registry suitable for performing the determined sub-goals, and

ordering a delegation of sub-goal requests complete the requested service; and

implementing the base goal satisfaction plan.

30. A computer program as recited in claim 29 wherein the computer executable instruction for providing an agent registry includes the following computer executable instructions for registering a specific service-providing electronic agent into the agents registry

establishing a bi-directional communication link between the specific agent and a facilitator agent controlling the agent registry;

providing a new agent profile to the facilitator agent, the new agent profile defining publicly available capabilities of the specific agent; and

registering the specific agent together with the new agent profile within the agent registry, thereby making available to the facilitator agent the capabilities of the specific agent.

31. A computer program as recited in claim 30 wherein the computer executable instruction for registering a specific agent further includes:

invoking the specific agent in order to activate the specific agent;

instating an instance of the specific agent; and

transmitting the new agent profile from the specific agent to the facilitator agent in response to the instantiation of the specific agent.

32. A computer program as recited in claim 29 wherein the computer executable instruction for providing an agent registry includes a computer executable instruction for removing a specific service-providing electronic agent from the registry upon determining that the specific agent is no longer available to provide services.

33. A computer program as recited in claim 29 wherein the provided agent registry includes a symbolic name, a unique address, data declarations, trigger declarations, task declarations, and process characteristics for each active agent.

34. Computer program as recited in claim 29 further including computer executable instructions for receiving the service request via a communications link established with a client.

35. A computer program as recited in claim 29 wherein the computer executable instruction for providing a service request includes instructions for:

receiving a non-ICL format service request;

US 6,851,115 B1

33

selecting an active agent capable of converting the non-ICL format service request into an ICL format service request;

forwarding the non-ICL format service request to the active agent capable of converting the non-ICL format service request, together with a request that such conversion be performed; and

receiving an ICL format service request corresponding to the non-ICL format service request.

36. A computer program as recited in claim 35 wherein the non-ICL format service request includes a natural language query, and the active agent capable of converting the non-ICL format service request into an ICL format service request is a natural language agent.

37. A computer program as recited in claim 36 wherein the natural language query is generated by a user in the agent.

38. A computer program as recited in claim 29, the computer program further including computer executable instructions for implementing a base goal that requires setting a trigger having conditional and consequential functionality.

39. A computer program as recited in claim 38 wherein the trigger is an outgoing communications trigger, the computer program further including computer executable instructions for:

monitoring all outgoing communication events in order to determine whether a specific outgoing communication event has occurred; and

in response to the occurrence of the specific outgoing communication event, performing the particular action defined by the trigger.

40. A computer program as recited in claim 38 wherein the trigger is an incoming communications trigger, the computer program further including computer executable instructions for:

monitoring all incoming communication events in order to determine whether a specific incoming communication event has occurred; and

in response to the occurrence of the specific incoming communication event, performing the particular action defined by the trigger.

41. A computer program as recited in claim 38 wherein the trigger is a data trigger, the computer program further including computer executable instructions for:

monitoring a state of a data repository; and

in response to a particular state event, performing the particular action defined by the trigger.

42. A computer program as recited in claim 38 wherein the trigger is a time trigger, the computer program further including computer executable instructions for:

monitoring for the occurrence of a particular time condition; and

in response to the occurrence of the particular time condition, performing the particular action defined by the trigger.

43. A computer program as recited in claim 38 further including computer executable instructions for instating and executing the trigger within the facilitator agent.

44. A computer program as recited in claim 38 further including computer executable instructions for instating and executing the trigger within a first service-providing agent.

45. A computer program as recited in claim 29 further including computer executable instructions for interpreting compound goals having sub-goals separated by operators.

34

46. A computer program as recited in claim 45 wherein the type of available operators includes a conjunction operator, a disjunction operator, and a conditional execution operator.

47. A computer program as recited in claim 46 wherein the type of available operators further includes parallel disjunction operator that indicates that distinct goals are to be performed by different agents.

48. An Interagent Communication Language (ICL) providing a basis for facilitated cooperative task completion within a distributed computing environment having a facilitator agent and a plurality of autonomous service-providing electronic agents, wherein:

the ICL having:

a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events; and

a content layer comprising one or more of goals, triggers and data elements associated with the events;

the ICL having one or more features from a set of features comprising:

enabling agents perform queries of other agents;

enabling agents to exchange information with other agents; and

enabling agents to set triggers within other agents; and

the ICL having a syntax supporting compound goal expressions wherein said compound goal expressions are such that goals within a single request provided according to the ICL syntax may be coupled by one or more operators from a set of operators comprising:

a conditional execution operator; and

a parallel disjunctive operation that indicates that disjunct goals are to be performed by different agents.

49. An ICL as recited in claim 48, wherein the ICL is computer platform independent.

50. An ICL as recited in claim 48 wherein the ICL is independent of computer programming languages which the plurality of agents are programmed in.

51. An ICL as recited in claim 48 wherein the ICL syntax supports explicit task completion constraints include use of specific agent constraints and response time constraints.

52. An ICL as recited in claim 51, wherein possible types of task completion constraints include use of specific agent constraints and response time constraints.

53. An ICL as recited in claim 51 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

54. An ICL as recited in claim 48 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

55. An ICL as recited in claim 48 wherein each autonomous service-providing electronic agent defines and publishes a set of capability declarations or solvables, expressed in ICL, that describes services provided by such electronic agent.

56. An ICL as recited in claim 55 wherein an electronic agent's solvables define an interface for the electronic agent.

57. An ICL as recited in claim 56 wherein the facilitator agent maintains an agent registry making available plurality of electronic agent interfaces.

58. An ICL as recited in claim 57 wherein the possible types of solvables includes procedure solvables, a procedure solvable operable to implement a procedure such as a test or an action.

59. An ICL as recited in claim 58 wherein the possible types of solvables further includes data solvables, a data solvable operable to provide access to a collection of data.

US 6,851,115 B1

35

60. An ICL as recited in claim 58 wherein the possible types of solvables includes data solvables, a data solvable operable to provide access to a collection of data.

61. A facilitator agent arranged to coordinate cooperative task completion within a distributed computing environment having a plurality of autonomous service-providing electronic agents, the facilitator agent comprising:

an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment; and

a facilitating engine operable to parse a service requesting order to interpret a compound goal set forth therein, the compound goal including both local and global constraints and control parameters, the service request formed according to an Interagent Communication Language (ICL), wherein the ICL includes:

a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events; and

a content layer comprising one or more of goals, triggers and data elements associated with the events; and

the facilitating engine further operable to construct a goal satisfaction plan by using reasoning that includes one or more of domain-independent coordination strategies, domain-specific reasoning, and application-specific reasoning comprising rules and learning algorithms.

62. A facilitator agent as recited in claim 61, wherein the facilitating engine is capable of modifying the goal satisfaction plan during execution, the modifying initiated by events such as new agent declarations within the agent registry, decisions made by remote agents, and information, provided to the facilitating engine by remote agents.

63. A facilitator agent as recited in claim 61 wherein the agent registry includes a symbolic name, a unique address, data declarations, trigger declarations, task declarations, and process characteristics for each active agent.

64. A facilitator agent as recited in claim 61 wherein the facilitating engine is operable to install a trigger mechanism requesting that a certain action be taken when a certain set of conditions are met.

65. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a communication trigger that monitors communication events and performs the certain action when a certain communication event occurs.

66. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a data trigger that monitors a state of a data repository and performs the certain action when a certain data state is obtained.

67. A facilitator agent as recited in claim 66 wherein the data repository is local to the facilitator agent.

68. A facilitator agent as recited in claim 66 wherein the data repository is remote from the facilitator agent.

69. A facilitator agent as recited in claim 64 wherein the trigger mechanism is a task trigger having a set of conditions.

70. A facilitator agent as recited in claim 61, the facilitator agent further including a global database accessible to at least one of the service-providing electronic agents.

71. A software-based, flexible computer architecture for communication and cooperation among distributed electronic agents, the architecture contemplating a distributed computing system comprising:

a plurality of service-providing electronic agents;

an Interagent Communication Language (ICL), wherein the inter-agent language includes:

36

a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events; and

a content layer comprising one or more of goals, triggers and data elements associated with the events; and

a facilitator agent in bi-directional communications with the plurality of service-providing electronic agents, the facilitator agent including:

an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment;

a facilitating engine operable to parse a service request in order to interpret an arbitrarily complex goal set forth therein, the facilitating engine further operable to construct a goal satisfaction plan including the coordination of a suitable delegation of sub-goal requests to best complete the requested service by using reasoning that includes one or more of domain-independent coordination strategies, domain-specific reasoning, and application-specific reasoning comprising rules and learning algorithms.

72. A computer architecture as recited in claim 71, wherein the Interagent Communication Language (ICL) is for enabling agents to perform queries of other agents, exchange information with other agents, and set triggers within other agents, the ICL further defined by an ICL syntax supporting compound goal expressions such that goals within single request provided according to the ICL syntax may be coupled by a conjunctive operator, a disjunctive operator, a conditional execution operator, and a parallel disjunctive operator parallel disjunctive operator that indicates that disjunct goals are to be performed by different agents.

73. A computer architecture as recited in claim 72, wherein the ICL is computer platform independent.

74. A computer architecture as recited in claim 73 wherein the ICL is independent of computer programming languages in which the plurality of agents are programmed.

75. A computer architecture as recited in claim 73 wherein the ICL syntax supports explicit task completion constraints within goal expressions.

76. A computer architecture as recited in claim 75 wherein possible types of task completion constraints include use of specific agent constraints and response time constraints.

77. A computer architecture as recited in claim 75 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

78. A computer architecture as recited in claim 73 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

79. A computer architecture as recited in claim 73 wherein each autonomous service-providing electronic agent defines and publishes a set of capability declarations or solvables, expressed in ICL, that describes services provided by such electronic agent.

80. A computer architecture as recited in claim 79 wherein an electronic agent's solvables define an interface for the electronic agent.

81. A computer architecture as recited in claim 80 wherein the possible types of solvables includes procedure solvables, a procedure solvable operable to implement a procedure such as a test or an action.

82. A computer architecture as recited in claim 81 wherein the possible types of solvables further includes data solvables, a data solvable operable to provide access to a collection of data.

US 6,851,115 B1

37

83. A computer architecture as recited in claim 82 wherein the possible types of solvables includes a data solvable operable to provide access to modify a collection of data.

84. A computer architecture as recited in claim 71 wherein a planning component of the facilitating engine are distributed across at least two computer processes.

85. A computer architecture as recited in claim 71 wherein an execution component of the facilitating engine is distributed across at least two computer process.

86. A data wave carrier providing a transport mechanism for information communication in a distributed computing environment having at least one facilitator agent and at least one active client agent, and an Interagent Communication Language (ICL), wherein the ICL includes:

a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events; and

a content layer comprising one or more of goals, triggers and data elements associated with the events; wherein said at least one facilitator agent is operable to construct a goal satisfaction plan by using reasoning that

38

includes one or more of domain-independent coordination strategies, domain-specific reasoning, and application-specific reasoning comprising rules and learning algorithms for satisfying one or more requests for service from said at least one active client agent, the data wave carrier comprising a signal representation of an inter-agent language description of an active client agent's functional capabilities.

87. A data wave carrier as recited in claim 86, the data wave carrier further comprising a corresponding signal representation of said one or more requests for service in the inter-agent language from a first agent to a second agent.

88. A data wave carrier as recited in claim 86, the data wave carrier further comprising a signal representation of a goal dispatched to an agent for performance from a facilitator agent.

89. A data wave carrier as recited in claim 88 wherein a later state of the data wave carrier comprises a signal representation of a response to the dispatched goal including results and/or a status report from the agent for performance to the facilitator agent.

* * * * *

Exhibit 2

U 8037680

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

March 11, 2020

**THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM
THE RECORDS OF THIS OFFICE OF:**

U.S. PATENT: 7,069,560

ISSUE DATE: June 27, 2006

**By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office**



R GLOVER
Certifying Officer



US007069560B1

(12) **United States Patent**
Cheyer et al.

(10) **Patent No.:** **US 7,069,560 B1**
(45) **Date of Patent:** ***Jun. 27, 2006**

(54) **HIGHLY SCALABLE SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS**

(75) Inventors: **Adam J. Cheyer**, Palo Alto, CA (US);
David L. Martin, Santa Clara, CA (US)

(73) Assignee: **SRI International**, Menlo Park, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

5,774,859 A	6/1998	Houser et al.	
5,794,050 A	8/1998	Dahlgren et al.	
5,802,526 A	9/1998	Fawcett et al.	
5,805,775 A	9/1998	Eberman et al.	
5,855,002 A	12/1998	Armstrong	
5,890,123 A	3/1999	Brown et al.	
5,963,940 A	10/1999	Liddy et al.	
6,003,072 A	12/1999	Gerritsen et al.	
6,012,030 A	1/2000	French-St. George et al.	
6,021,427 A	2/2000	Spagna et al.	
6,026,388 A	2/2000	Liddy et al.	
6,049,819 A *	4/2000	Buckle et al.	709/202
6,080,202 A	6/2000	Strickland et al.	
6,134,580 A *	10/2000	Tahara et al.	709/202
6,144,989 A	11/2000	Hodjat et al.	
6,173,279 B1	1/2001	Levin et al.	
6,192,338 B1	2/2001	Haszto et al.	

(Continued)

FOREIGN PATENT DOCUMENTS

EP 0 803 826 A2 10/1997

(Continued)

OTHER PUBLICATIONS

Nwana, Hyacinth et al. "Software Agent Technologies". BT Technology Journal. 1996.*

(Continued)

Primary Examiner—Lewis A. Bullock, Jr.

(74) Attorney, Agent, or Firm—Perkins Coie LLP

(57) **ABSTRACT**

A highly flexible, software-based architecture is disclosed for supporting cooperative task completion by flexible, dynamic configurations of autonomous electronic agents. Communications and cooperation between agents are brokered by one or more facilitators. The facilitators employ strategic reasoning for generating a goal satisfaction plan to fulfill arbitrarily complex goals by users and service requesting agents.

55 Claims, 16 Drawing Sheets

(21) Appl. No.: **09/271,614**

(22) Filed: **Mar. 17, 1999**

Related U.S. Application Data

(63) Continuation of application No. 09/225,198, filed on Jan. 5, 1999, now Pat. No. 6,851,115.

(51) Int. Cl. **G06F 9/54** (2006.01)

(52) U.S. Cl. 719/317; 709/202

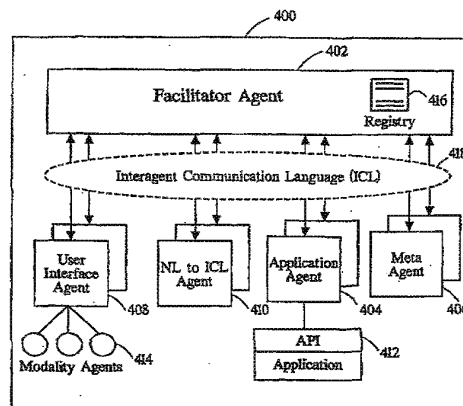
(58) Field of Classification Search 709/317, 709/202; 719/317

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,197,005 A	3/1993	Schwartz et al.
5,386,556 A	1/1995	Hedin et al.
5,434,777 A	7/1995	Luciw
5,519,608 A	5/1996	Kupiec
5,608,624 A	3/1997	Luciw
5,721,938 A	2/1998	Stuckey
5,729,659 A	3/1998	Potter
5,748,974 A	5/1998	Johnson



US 7,069,560 B1

Page 2

U.S. PATENT DOCUMENTS

6,226,666	B1	5/2001	Chang et al.	
6,338,081	B1	1/2002	Furusawa et al.	
6,484,155	B1 *	11/2002	Kiss et al.	706/46
6,557,025	B1 *	4/2003	Hattori et al.	709/202
2001/0039562	A1 *	11/2001	Sato	709/202
2003/0167247	A1 *	9/2003	Masuoka	706/46

FOREIGN PATENT DOCUMENTS

WO	WO 00/11869	3/2000
----	-------------	--------

OTHER PUBLICATIONS

Busetta, Paolo et al. "The BDIM Agent Toolkit Design." 1997.*

Mayfield, James et al. "Desiderata for Agent Communication Languages." Mar. 27-29, 1995.*

Khedro, Taha et al. Concurrent Engineering through Interoperable Software Agents. Aug. 1994.*

Dowding, John et al., "Gemini: A Natural Language System For Spoken-Language Understanding", SRI International. <http://www.ai.sri.com/~oaa/infowiz.html>, InfoWiz: An Animated Voice Interactive Information System, May 8, 2000.

Dowding, John, "Interleaving Syntax and Semantics in an Efficient Bottom-up Parser", SRI International.

Moore, Robert et al., "Combining Linguistic and Statistical Knowledge Sources in a Natural-Language Processing for ATIS", SRI International.

Stent, Amanda et al., "The CommandTalk Spoken Dialog System", SRI International.

Moore, Robert et al., CommandTalk: A Spoken-Language Interface for Battlefield Simulations:, Oct. 23, 1997, SRI International.

Dowding, John et al., "Interpreting Language in Context in CommandTalk", Feb. 5, 1999, SRI International.

Moran, Douglas B. and Cheyer, Adam J., "Intelligent Agent-based User Interfaces", Article Intelligence center, SRI International.

Martin, David L., Cheyer, Adam J. and Moran, Douglas B., "Building Distributed Software Systems with the Open Agent Architecture".

Cohen, Philip R. and Cheyer, Adam, SRI International, Wang, Michelle, Stanford University, Baeg, Soon Cheol, Etri, "An Open Agent Architecture" AAAI Spring Symposium, pp1-8, Mar. 1994.

Julia, Luc E. and Cheyer, Adam J., SRI International "Cooperative Agents and Recognition Systems (CARS) for Drivers and Passengers".

Moran, Douglas B., Cheyer, Adam J., Julia, Luc E., Martin, David L., SRI International, and Park, Sangkyu, Electronics and Telecommunications Research Institute, "Multimodal User Interfaces in the Open Agent Architecture".

Chey, Adam and Lulia, Luc, SRI International "Multimodal Maps: An Agent-based Approach".

Cutkosky, Mark R., Engelm, Robert S., Fikes, Richard E., Genesereth, Michael R., Gruber, Thomas R., Stanford University, Mark, William, Lockheed Palo Alto Research Labs, Tenenbaum, Jay M., Weber, Jay C., Enterprise Integration Technologies, "An Experiment in Integrating Concurrent Engineering Systems".

Martin, David L., Cheyer, Adam, SRI International, Lee, Gowang-Lo, Etri, "Development Tools for the Open Agent Architecture", The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96), London, Apr. 1996.

Chey, Adam, Martin, David and Moran, Douglas, "The Open Agent architecture™", SRI International, AI Center. Dejima, Inc., <http://www.dejima.com/>.

Martin, David; Oohama, Hiroki; Moran, Douglas; Cheyer, Adam; "Information Brokering in an Agent Architecture," Proceeding of the 2nd International Conference on Practical Application of Intelligent Agents & Multi-Agent Technology, London, Apr. 1997.

* cited by examiner

U.S. Patent

Jun. 27, 2006

Sheet 1 of 16

US 7,069,560 B1

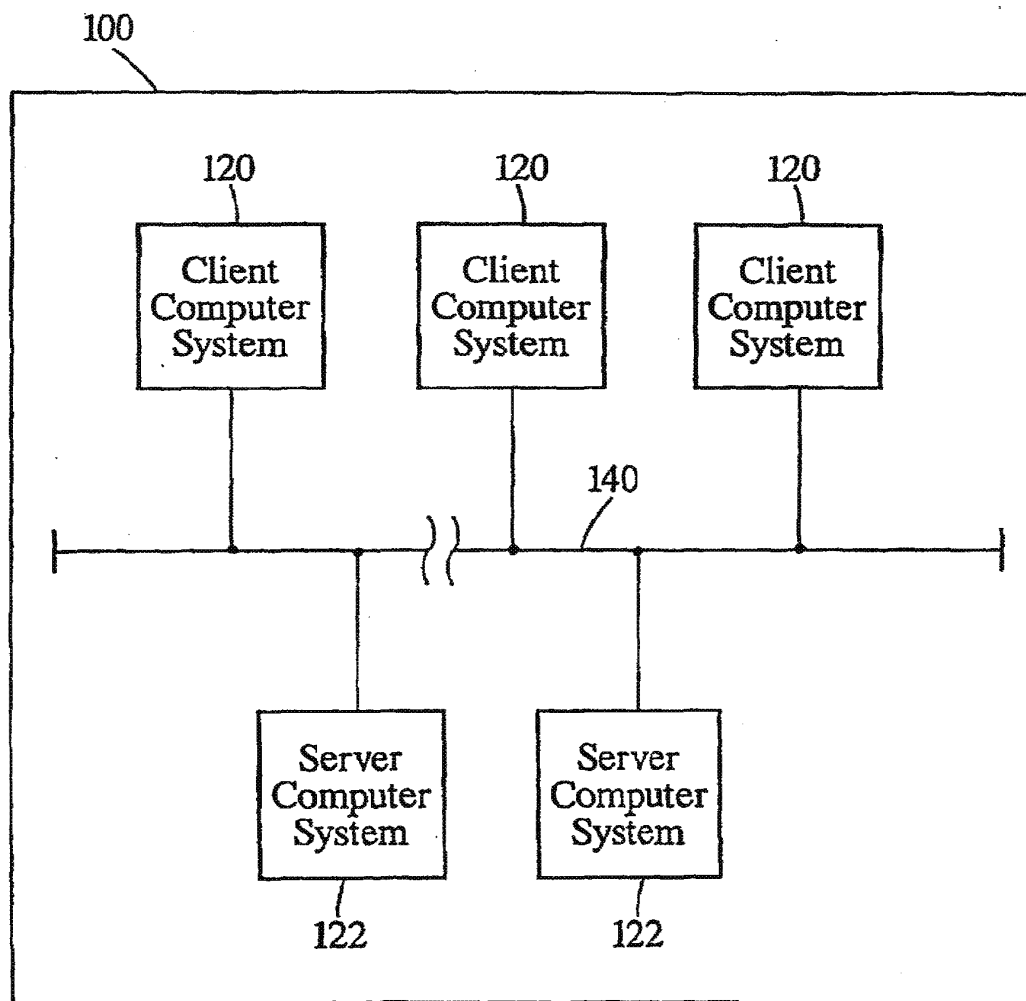


Fig. 1
(Prior Art)

U.S. Patent

Jun. 27, 2006

Sheet 2 of 16

US 7,069,560 B1

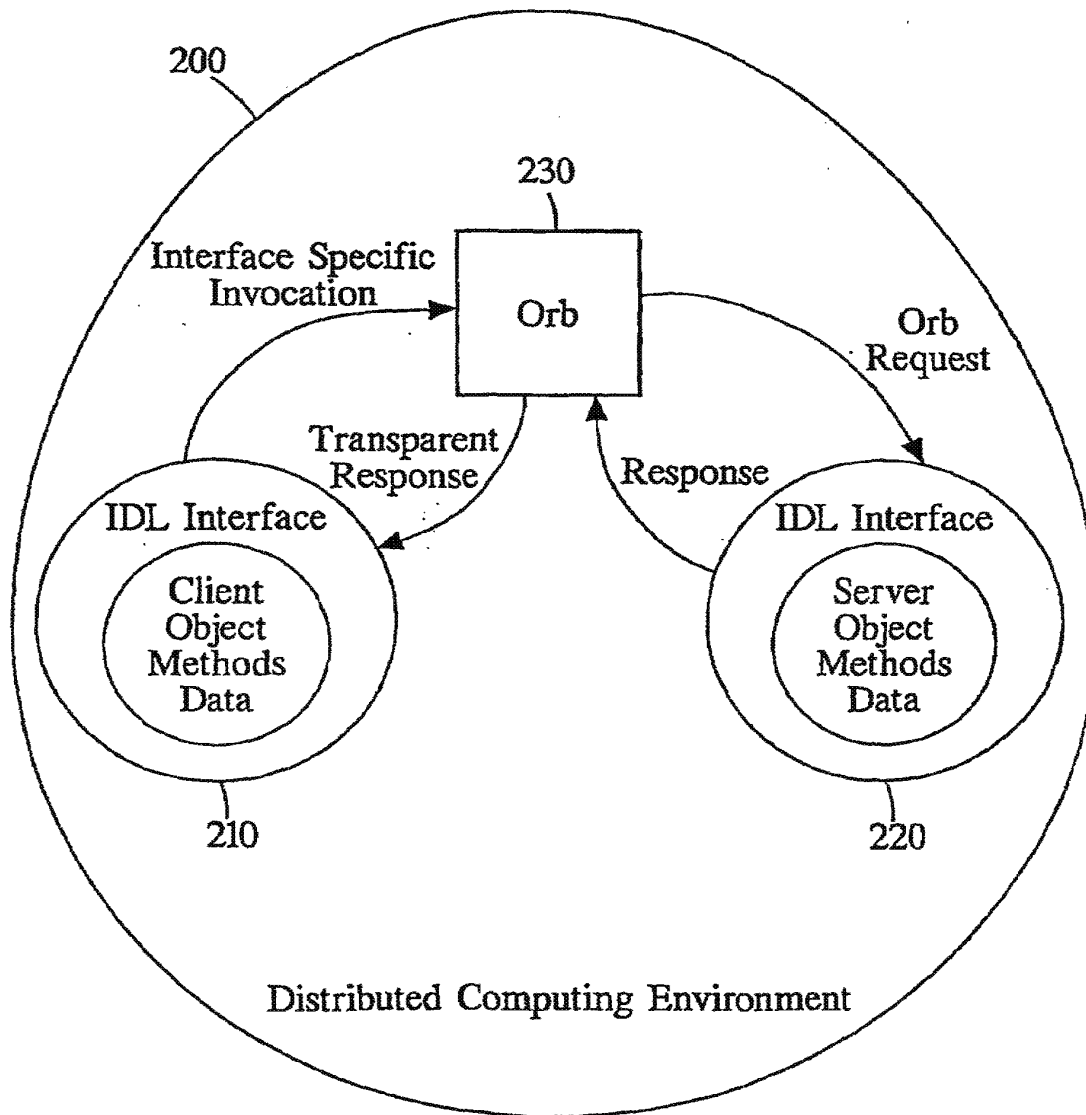


Fig. 2
(Prior Art)

U.S. Patent

Jun. 27, 2006

Sheet 3 of 16

US 7,069,560 B1

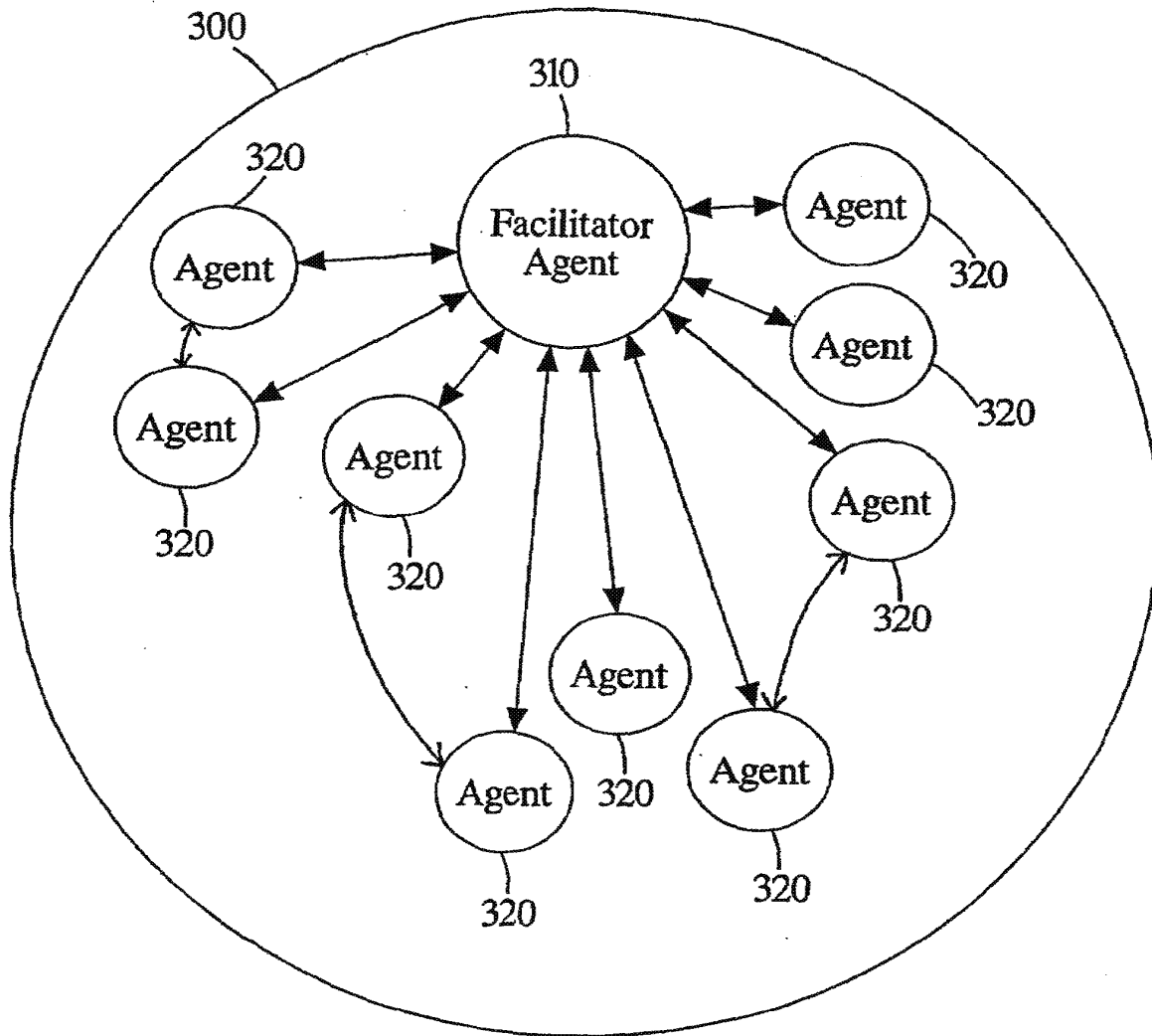


Fig. 3

U.S. Patent

Jun. 27, 2006

Sheet 4 of 16

US 7,069,560 B1

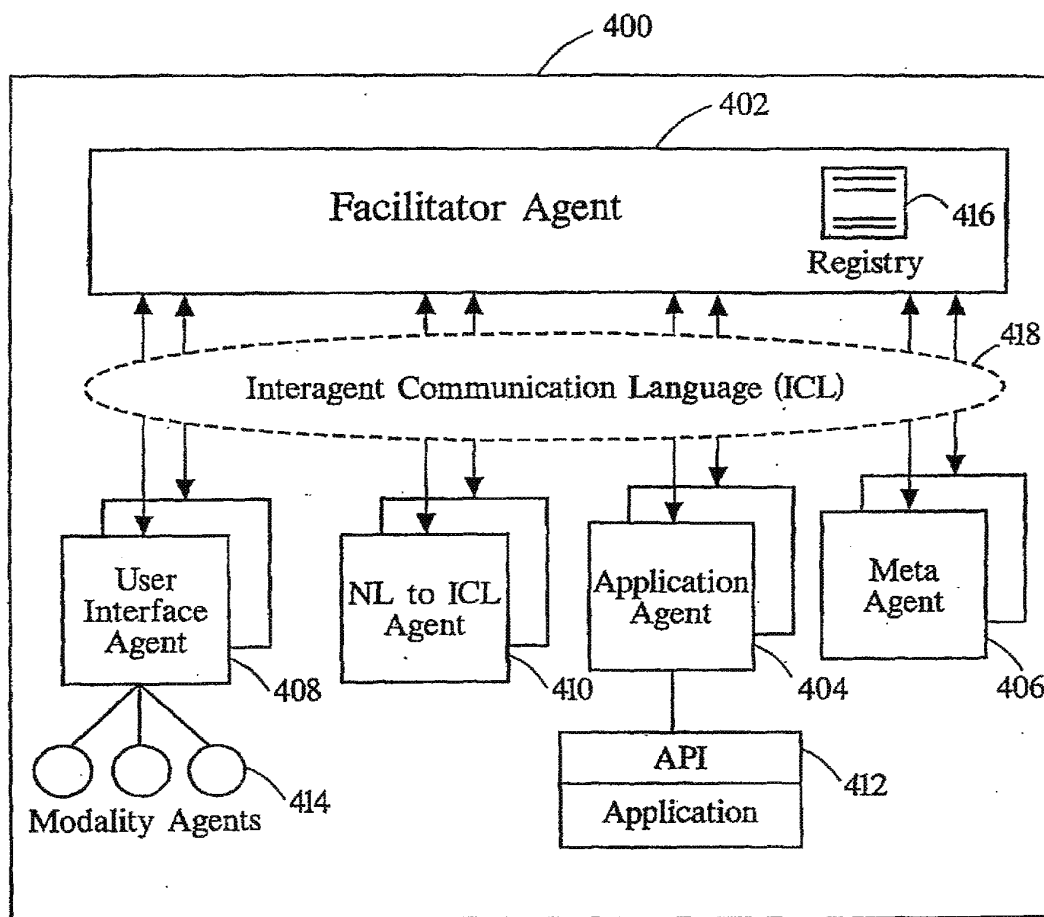


Fig. 4

U.S. Patent

Jun. 27, 2006

Sheet 5 of 16

US 7,069,560 B1

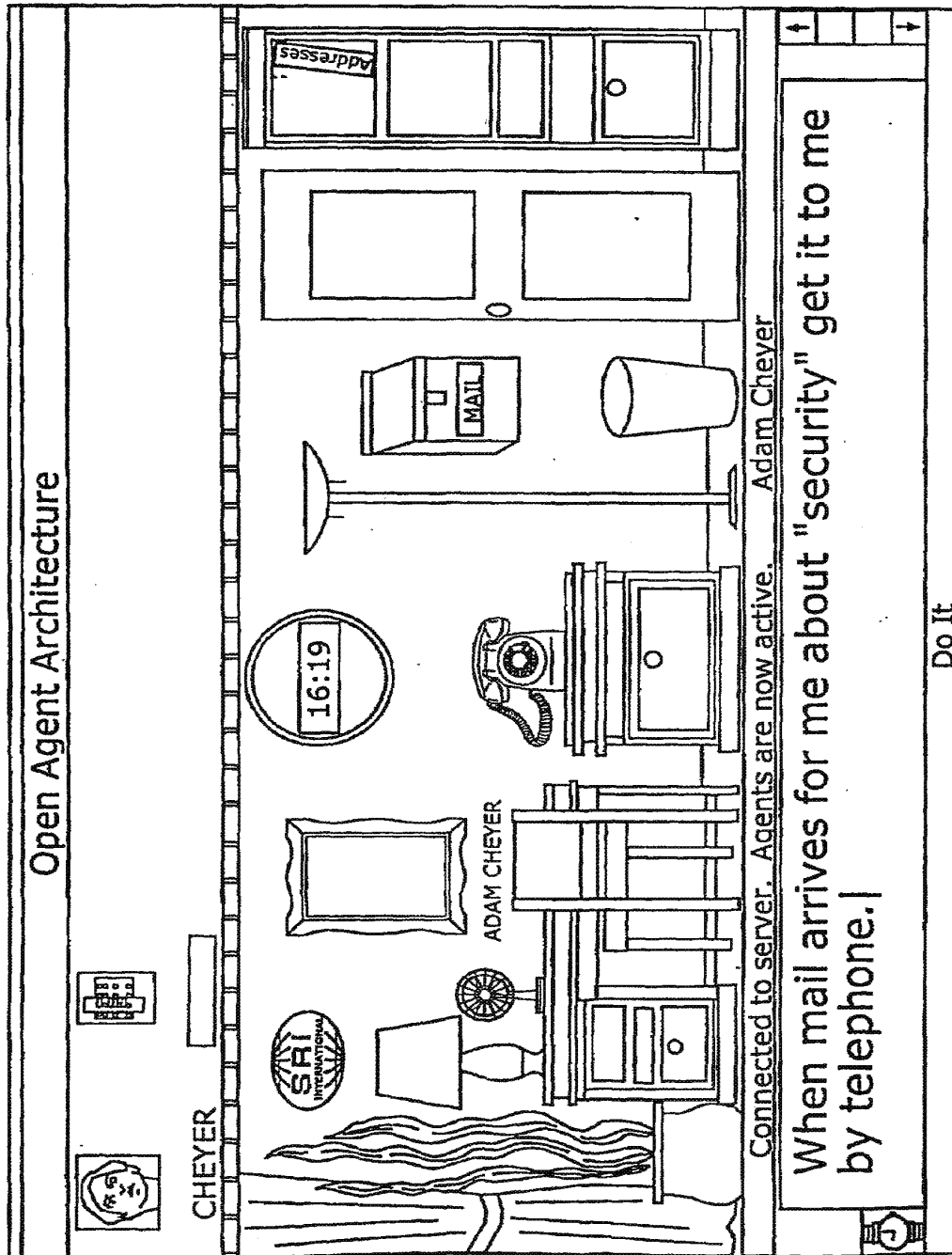


Fig. 5

U.S. Patent

Jun. 27, 2006

Sheet 6 of 16

US 7,069,560 B1

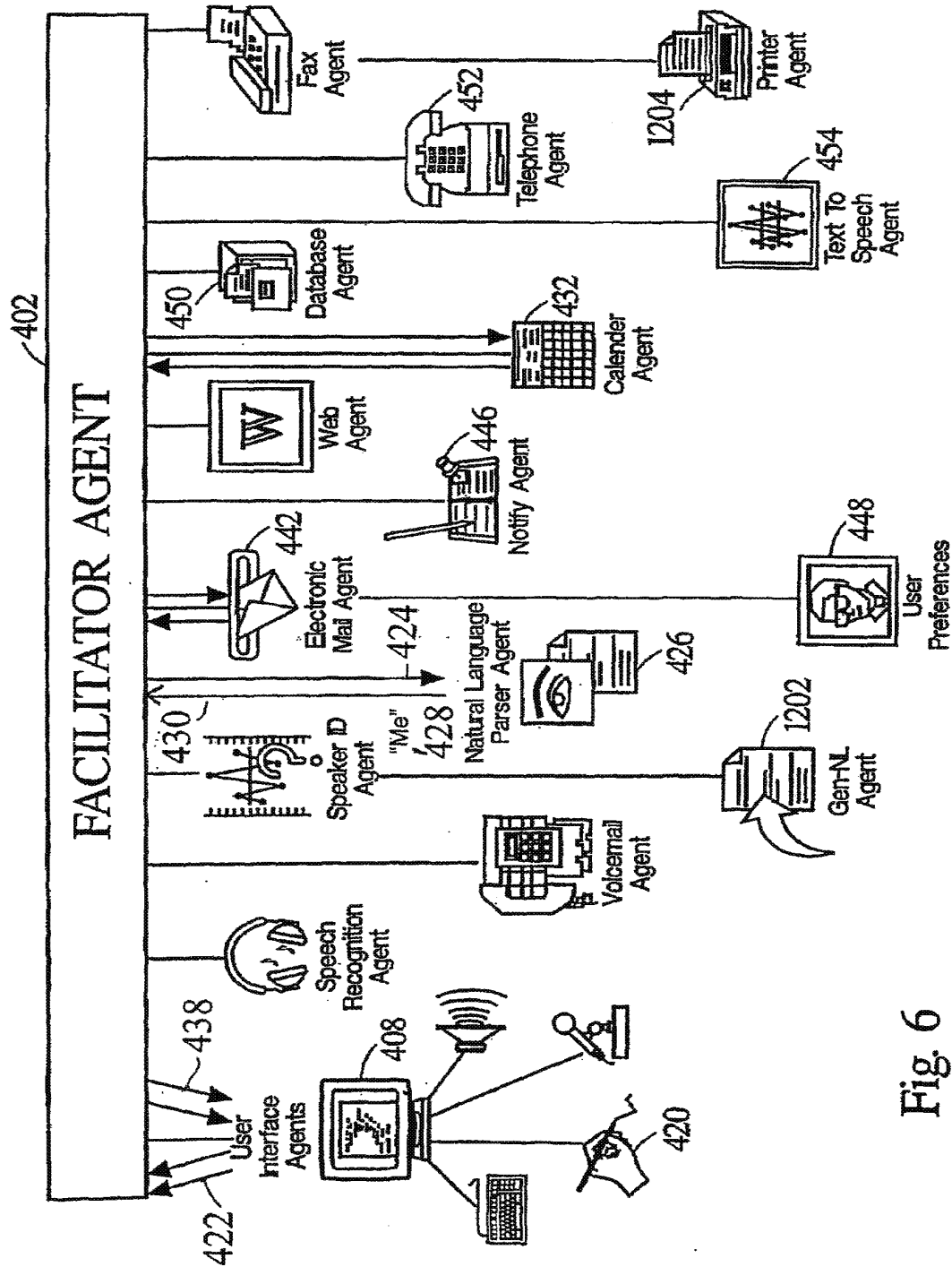


Fig. 6

U.S. Patent

Jun. 27, 2006

Sheet 7 of 16

US 7,069,560 B1

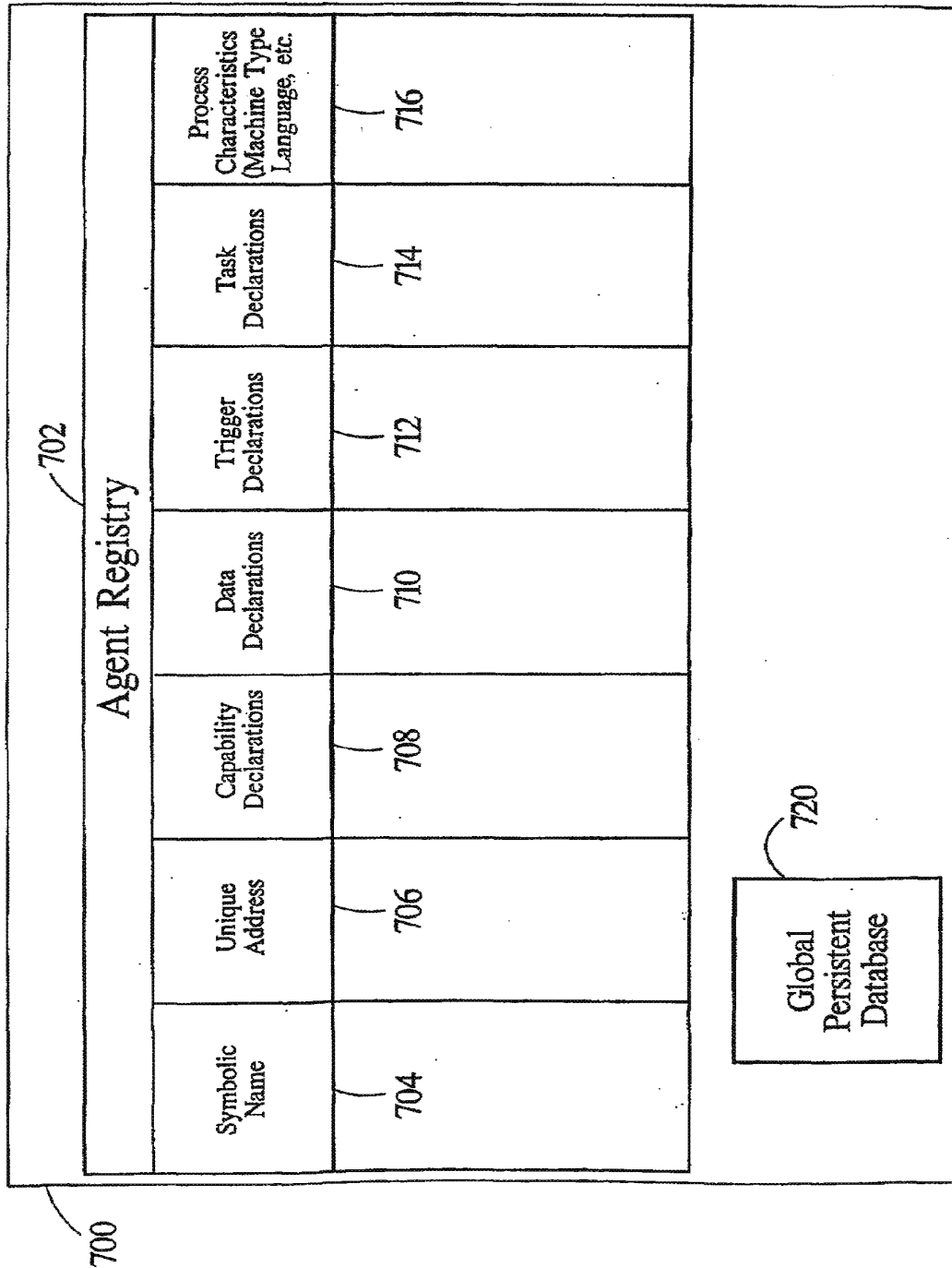


Fig. 7

U.S. Patent

Jun. 27, 2006

Sheet 8 of 16

US 7,069,560 B1

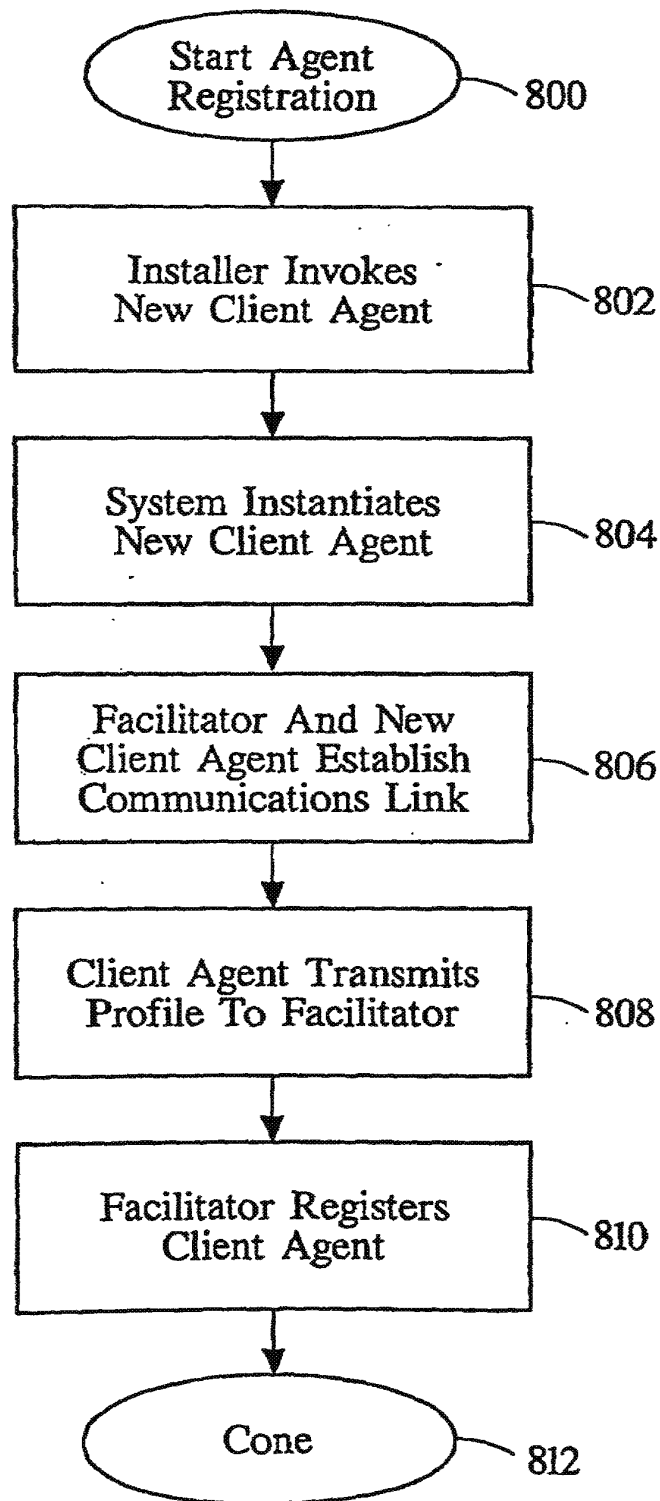


Fig. 8

U.S. Patent

Jun. 27, 2006

Sheet 9 of 16

US 7,069,560 B1

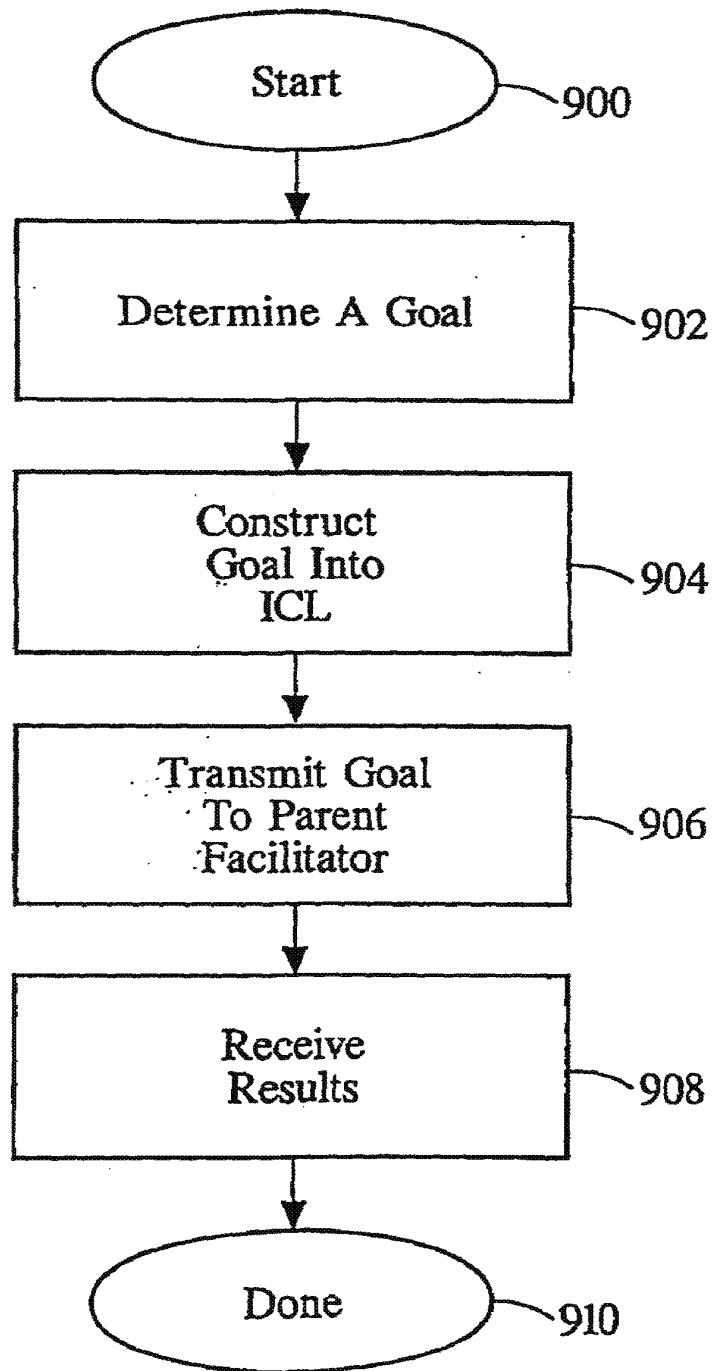


Fig. 9

U.S. Patent

Jun. 27, 2006

Sheet 10 of 16

US 7,069,560 B1

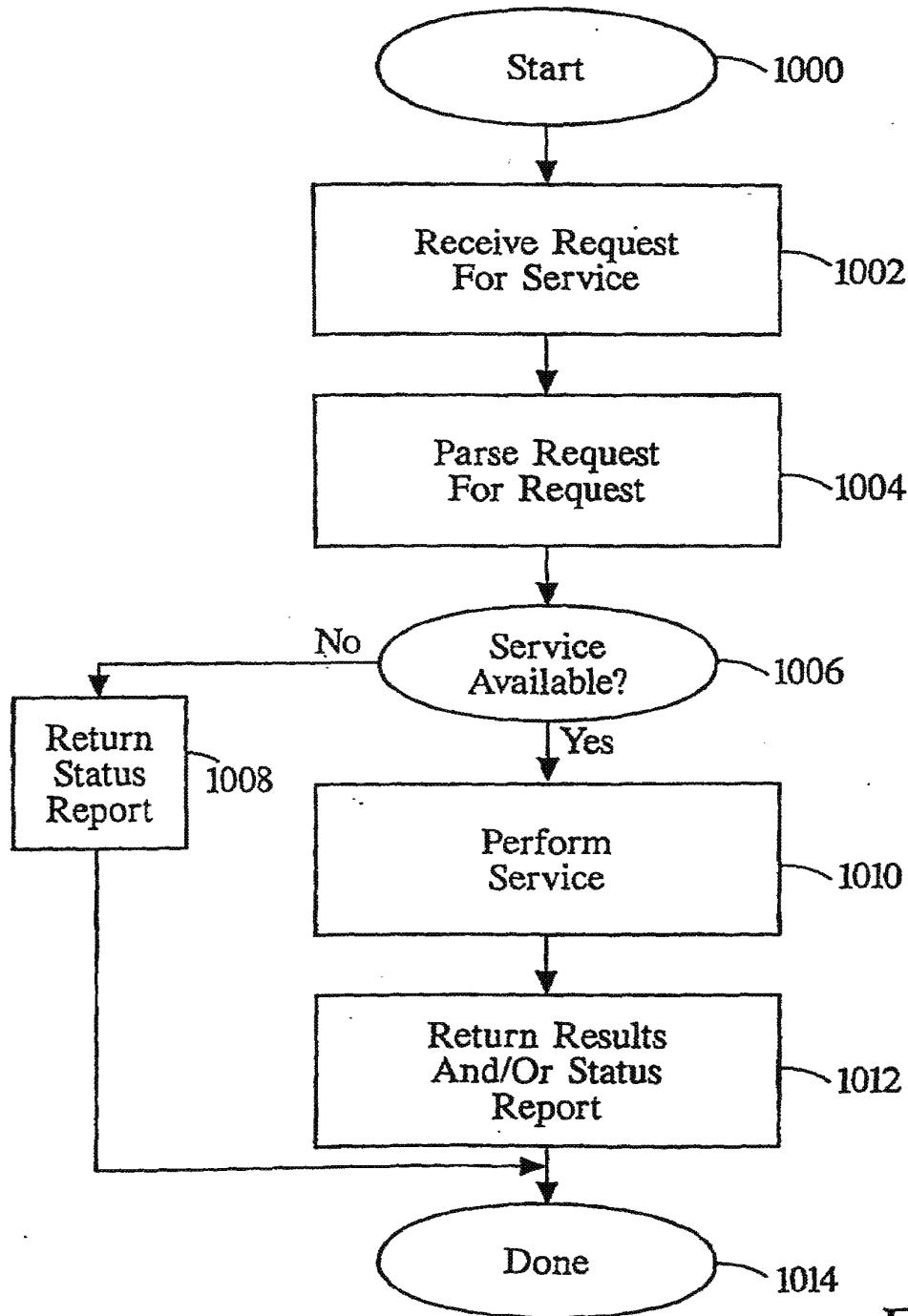


Fig. 10

U.S. Patent

Jun. 27, 2006

Sheet 11 of 16

US 7,069,560 B1

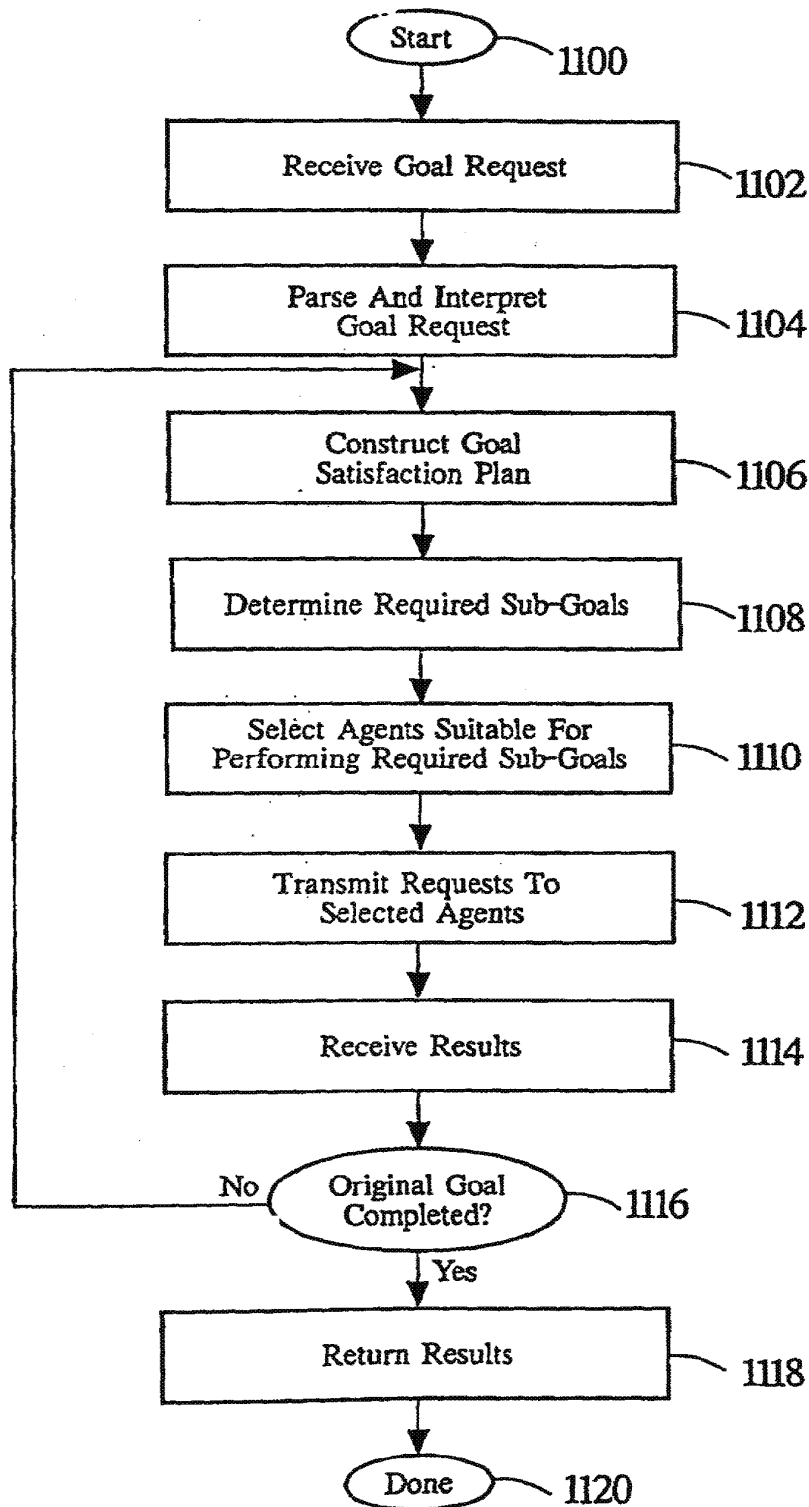


Fig. 11

U.S. Patent

Jun. 27, 2006

Sheet 12 of 16

US 7,069,560 B1

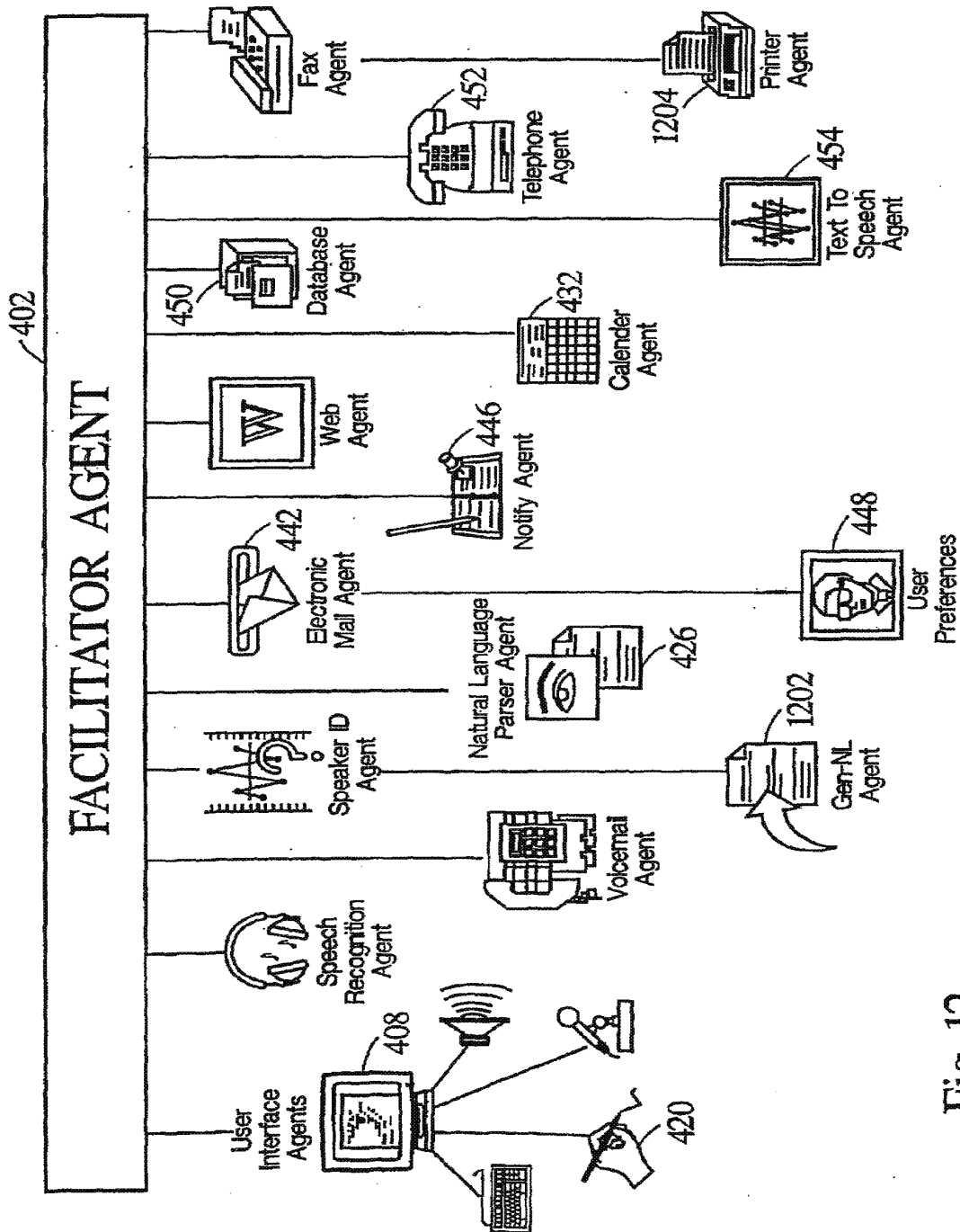


Fig. 12

U.S. Patent

Jun. 27, 2006

Sheet 13 of 16

US 7,069,560 B1

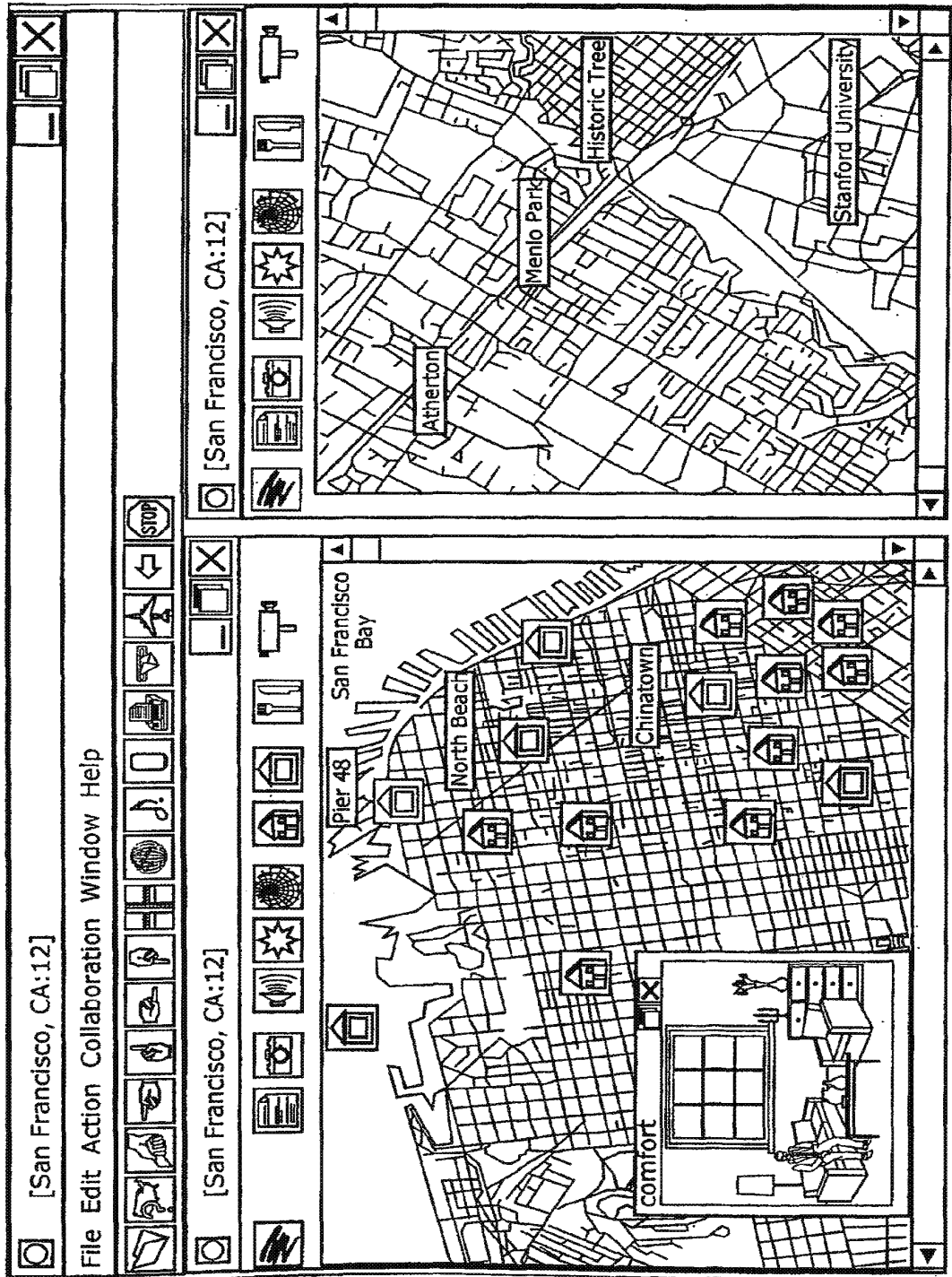


Fig. 13

U.S. Patent

Jun. 27, 2006

Sheet 14 of 16

US 7,069,560 B1

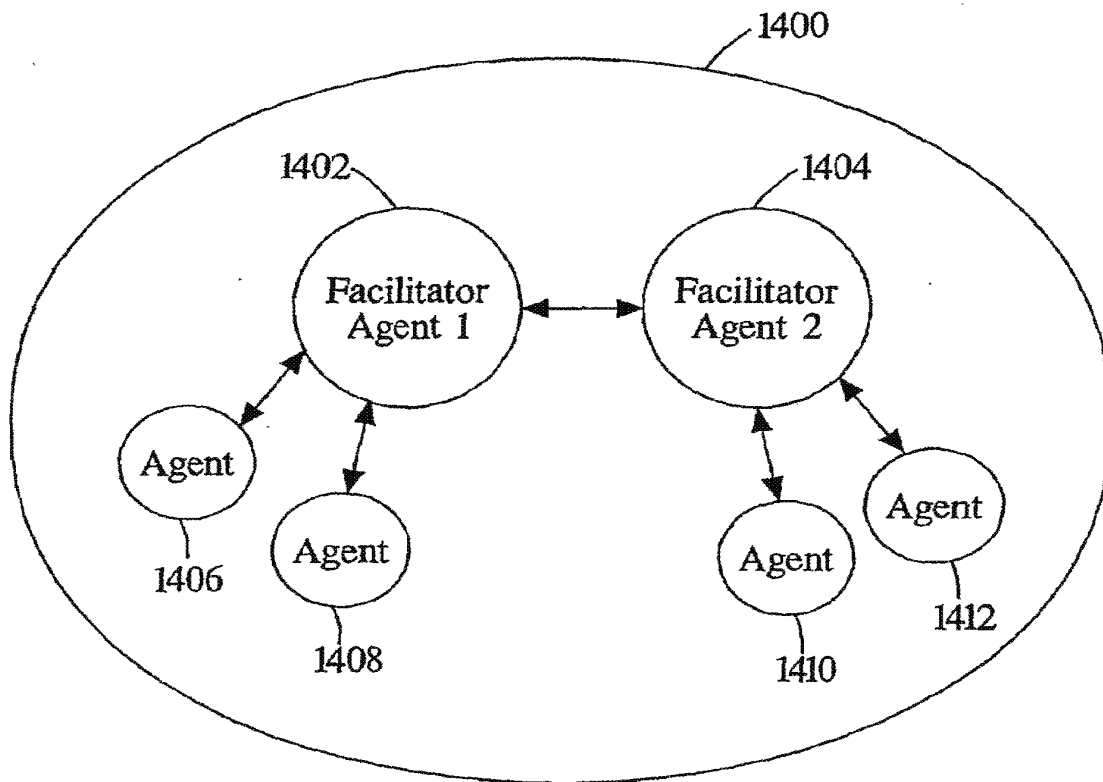


Fig. 14

U.S. Patent

Jun. 27, 2006

Sheet 15 of 16

US 7,069,560 B1

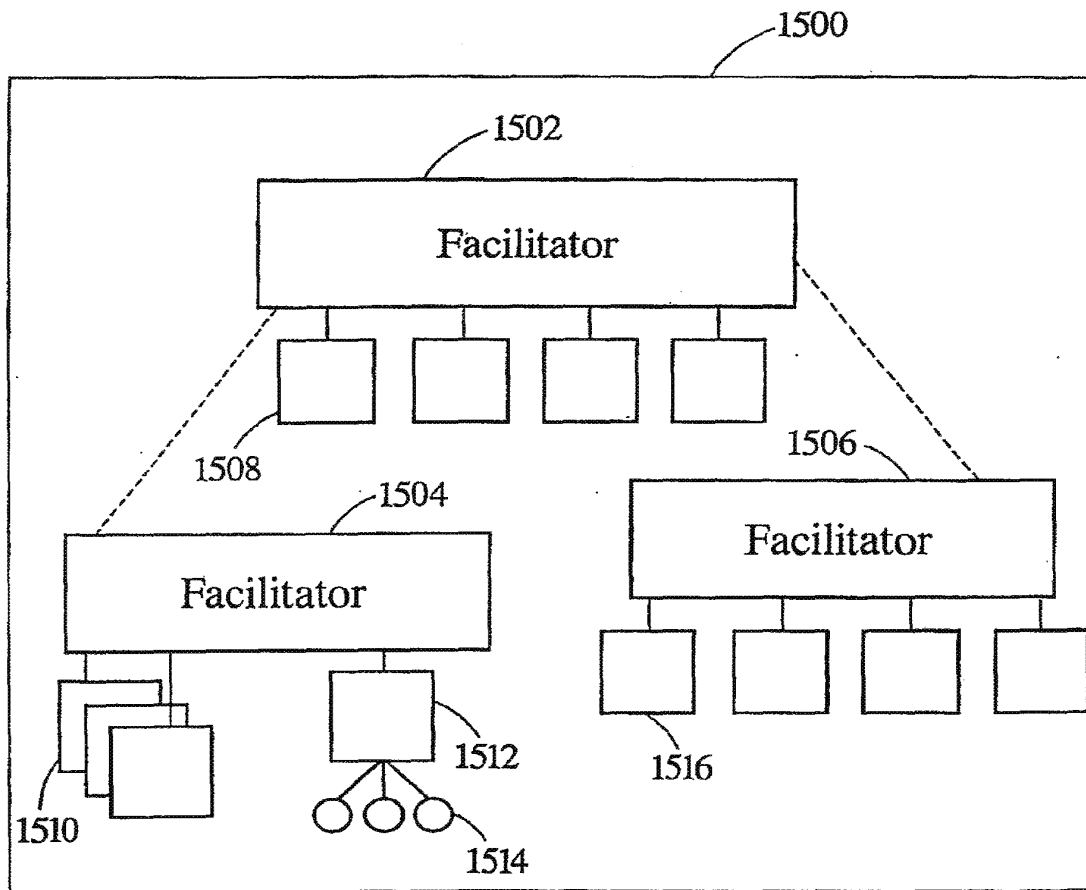


Fig. 15

U.S. Patent

Jun. 27, 2006

Sheet 16 of 16

US 7,069,560 B1

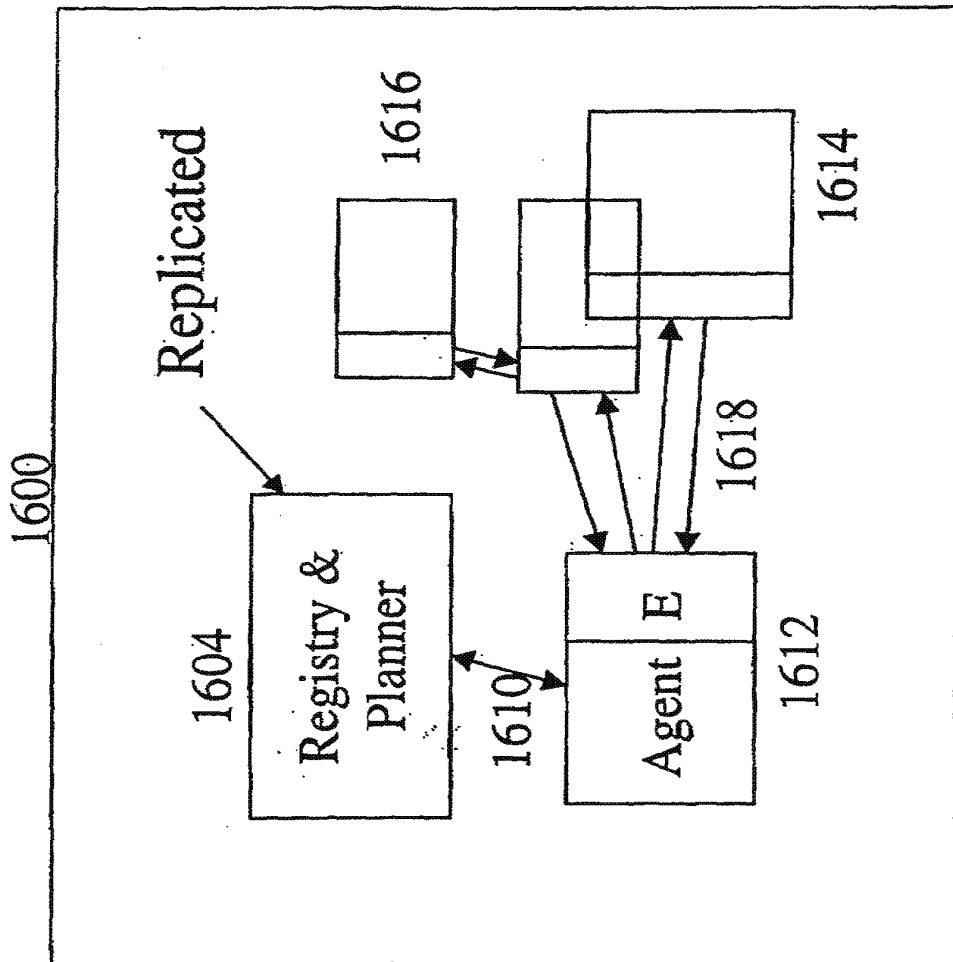


Figure 16

US 7,069,560 B1

1

HIGHLY SCALABLE SOFTWARE-BASED ARCHITECTURE FOR COMMUNICATION AND COOPERATION AMONG DISTRIBUTED ELECTRONIC AGENTS

CROSS-REFERENCE TO RELATED APPLICATIONS

This is a Continuation application of U.S. patent application Ser. No. 09/225,198, filed Jan. 5, 1999, now U.S. Pat. No. 6,851,115, which application is incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention is related to distributed computing environments and the completion of tasks within such environments. In particular, the present invention teaches a variety of software-based architectures for communication and cooperation among distributed electronic agents. Certain embodiments teach interagent communication languages enabling client agents to make requests in the form of arbitrarily complex goal expressions that are solved through facilitation by a facilitator agent.

Context and Motivation for Distributed Software Systems

The evolution of models for the design and construction of distributed software systems is being driven forward by several closely interrelated trends: the adoption of a networked computing model, rapidly rising expectations for smarter, longer-lived, more autonomous software applications and an ever increasing demand for more accessible and intuitive user interfaces.

Prior Art FIG. 1 illustrates a networked computing model 100 having a plurality of client and server computer systems 120 and 122 coupled together over a physical transport mechanism 140. The adoption of the networked computing model 100 has lead to a greatly increased reliance on distributed sites for both data and processing resources. Systems such as the networked computing model 100 are based upon at least one physical transport mechanism 140 coupling the multiple computer systems 120 and 122 to support the transfer of information between these computers. Some of these computers basically support using the network and are known as client computers (clients). Some of these computers provide resources to other computers and are known as server computers (servers). The servers 122 can vary greatly in the resources they possess, access they provide and services made available to other computers across a network. Servers may service other servers as well as clients.

The Internet is a computing system based upon this network computing model. The Internet is continually growing, stimulating a paradigm shift for computing away from requiring all relevant data and programs to reside on the user's desktop machine. The data now routinely accessed from computers spread around the world has become increasingly rich in format, comprising multimedia documents, and audio and video streams. With the popularization of programming languages such as JAVA, data transported between local and remote machines may also include programs that can be downloaded and executed on the local machine. There is an ever increasing reliance on networked computing, necessitating software design approaches that allow for flexible composition of distributed processing elements in a dynamically changing and relatively unstable environment.

2

In an increasing variety of domains, application designers and users are coming to expect the deployment of smarter, longer-lived, more autonomous, software applications. Push technology, persistent monitoring of information sources, and the maintenance of user models, allowing for personalized responses and sharing of preferences, are examples of the simplest manifestations of this trend. Commercial enterprises are introducing significantly more advanced approaches, in many cases employing recent research results from artificial intelligence, data mining, machine learning, and other fields.

More than ever before, the increasing complexity of systems, the development of new technologies, and the availability of multimedia material and environments are creating a demand for more accessible and intuitive user interfaces. Autonomous, distributed, multi-component systems providing sophisticated services will no longer lend themselves to the familiar "direct manipulation" model of interaction, in which an individual user masters a fixed selection of commands provided by a single application. Ubiquitous computing, in networked environments, has brought about a situation in which the typical user of many software services is likely to be a non-expert, who may access a given service infrequently or only a few times. Accommodating such usage patterns calls for new approaches. Fortunately, input modalities now becoming widely available, such as speech recognition and pen-based handwriting/gesture recognition, and the ability to manage the presentation of systems' responses by using multiple media provide an opportunity to fashion a style of human-computer interaction that draws much more heavily on our experience with human—human interactions.

PRIOR RELATED ART

Existing approaches and technologies for distributed computing include distributed objects, mobile objects, blackboard-style architectures, and agent-based software engineering.

The Distributed Object Approach

Object-oriented languages, such as C++ or JAVA, provide significant advances over standard procedural languages with respect to the reusability and modularity of code: encapsulation, inheritance and polymorphism. Encapsulation encourages the creation of library interfaces that minimize dependencies on underlying algorithms or data structures. Changes to programming internals can be made at a later date with requiring modifications to the code that uses the library. Inheritance permits the extension and modification of a library of routines and data without requiring source code to the original library. Polymorphism allows one body of code to work on an arbitrary number of data types. For the sake of simplicity traditional objects may be seen to contain both methods and data. Methods provide the mechanisms by which the internal state of an object may be modified or by which communication may occur with another object or by which the instantiation or removal of objects may be directed.

With reference to FIG. 2, a distributed object technology based around an Object Request Broker will now be described. Whereas "standard" object-oriented programming (OOP) languages can be used to build monolithic programs out of many object building blocks, distributed object technologies (DOOP) allow the creation of programs whose components may be spread across multiple machines. As shown in FIG. 2, an object system 200 includes client

US 7,069,560 B1

3

objects 210 and server objects 220. To implement a client-server relationship between objects, the distributed object system 200 uses a registry mechanism (CORBA's registry is called an Object Request Broker, or ORB) 230 to store the interface descriptions of available objects. Through the services of the ORB 230, a client can transparently invoke a method on a remote server object. The ORB 230 is then responsible for finding the object 220 that can implement the request, passing it the parameters, invoking its method, and returning the results. In the most sophisticated systems, the client 210 does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of the server object's interface. Although distributed objects offer a powerful paradigm for creating networked applications, certain aspects of the approach are not perfectly tailored to the constantly changing environment of the Internet. A major restriction of the DOOP approach is that the interactions among objects are fixed through explicitly coded instructions by the application developer. It is often difficult to reuse an object in a new application without bringing along all its inherent dependencies on other objects (embedded interface definitions and explicit method calls). Another restriction of the DOOP approach is the result of its reliance on a remote procedure call (RPC) style of communication. Although easy to debug, this single thread of execution model does not facilitate programming to exploit the potential for parallel computation that one would expect in a distributed environment. In addition, RPC uses a blocking (synchronous) scheme that does not scale well for high-volume transactions.

Mobile Objects

Mobile objects, sometimes called mobile agents, are bits of code that can move to another execution site (presumably on a different machine) under their own programmatic control, where they can then interact with the local environment. For certain types of problems, the mobile object paradigm offers advantages over more traditional distributed object approaches. These advantages include network bandwidth and parallelism. Network bandwidth advantages exist for some database queries or electronic commerce applications, where it is more efficient to perform tests on data by bringing the tests to the data than by bringing large amounts of data to the testing program. Parallelism advantages include situations in which mobile agents can be spawned in parallel to accomplish many tasks at once.

Some of the disadvantages and inconveniences of the mobile agent approach include the programmatic specificity of the agent interactions, lack of coordination support between participant agents and execution environment irregularities regarding specific programming languages supported by host processors upon which agents reside. In a fashion similar to that of DOOP programming, an agent developer must programmatically specify where to go and how to interact with the target environment. There is generally little coordination support to encourage interactions among multiple (mobile) participants. Agents must be written in the programming language supported by the execution environment, whereas many other distributed technologies support heterogeneous communities of components, written in diverse programming languages.

Blackboard Architectures

Blackboard architectures typically allow multiple processes to communicate by reading and writing tuples from a global data store. Each process can watch for items of interest, perform computations based on the state of the

4

blackboard, and then add partial results or queries that other processes can consider. Blackboard architectures provide a flexible framework for problem solving by a dynamic community of distributed processes. A blackboard architecture provides one solution to eliminating the tightly bound interaction links that some of the other distributed technologies require during interprocess communication. This advantage can also be a disadvantage: although a programmer does not need to refer to a specific process during computation, the framework does not provide programmatic control for doing so in cases where this would be practical.

Agent-Based Software Engineering

Several research communities have approached distributed computing by casting it as a problem of modeling communication and cooperation among autonomous entities, or agents. Effective communication among independent agents requires four components: (1) a transport mechanism carrying messages in an asynchronous fashion, (2) an interaction protocol defining various types of communication interchange and their social implications (for instance, a response is expected of a question), (3) a content language permitting the expression and interpretation of utterances, and (4) an agreed-upon set of shared vocabulary and meaning for concepts (often called an ontology). Such mechanisms permit a much richer style of interaction among participants than can be expressed using a distributed object's RPC model or a blackboard architecture's centralized exchange approach.

Agent-based systems have shown much promise for flexible, fault-tolerant, distributed problem solving. Several agent-based projects have helped to evolve the notion of facilitation. However, existing agent-based technologies and architectures are typically very limited in the extent to which agents can specify complex goals or influence the strategies used by the facilitator. Further, such prior systems are not sufficiently attuned to the importance of integrating human agents (i.e., users) through natural language and other human-oriented user interface technologies.

The initial version of SRI International's Open Agent Architecture™ ("OAA®") technology provided only a very limited mechanism for dealing with compound goals. Fixed formats were available for specifying a flat list of either conjoined (AND) sub-goals or disjoined (OR) sub-goals; in both cases, parallel goal solving was hardwired in, and only a single set of parameters for the entire list could be specified. More complex goal expressions involving (for example) combinations of different boolean connectors, nested expressions, or conditionally interdependent ("IF... THEN") goals were not supported. Further, system scalability was not adequately addressed in this prior work.

SUMMARY OF INVENTION

A first embodiment of the present invention discloses a highly flexible, software-based architecture for constructing distributed systems. The architecture supports cooperative task completion by flexible, dynamic configurations of autonomous electronic agents. Communication and cooperation between agents are brokered by one or more facilitators, which are responsible for matching requests, from users and agents, with descriptions of the capabilities of other agents. It is not generally required that a user or agent know the identities, locations, or number of other agents involved in satisfying a request, and relatively minimal effort is involved in incorporating new agents and "wrapping" legacy applications. Extreme flexibility is achieved

US 7,069,560 B1

5

through an architecture organized around the declaration of capabilities by service-providing agents, the construction of arbitrarily complex goals by users and service-requesting agents, and the role of facilitators in delegating and coordinating the satisfaction of these goals, subject to advice and constraints that may accompany them. Additional mechanisms and features include facilities for creating and maintaining shared repositories of data; the use of triggers to instantiate commitments within and between agents; agent-based provision of multi-modal user interfaces, including natural language; and built-in support for including the user as a privileged member of the agent community. Specific embodiments providing enhanced scalability are also described.

BRIEF DESCRIPTION OF THE DRAWINGS

Prior Art

Prior Art FIG. 1 depicts a networked computing model;
Prior Art FIG. 2 depicts a distributed object technology based around an Object Resource Broker;

Examples of the Invention

FIG. 3 depicts a distributed agent system based around a facilitator agent;

FIG. 4 presents a structure typical of one small system of the present invention;

FIG. 5 depicts an Automated Office system implemented in accordance with an example embodiment of the present invention supporting a mobile user with a laptop computer and a telephone;

FIG. 6 schematically depicts an Automated Office system implemented as a network of agents in accordance with a preferred embodiment of the present invention;

FIG. 7 schematically shows data structures internal to a facilitator in accordance with a preferred embodiment of the present invention;

FIG. 8 depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention;

FIG. 9 depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a certain preferred embodiment of the present invention;

FIG. 10 depicts operations involved in a client agent responding to a service request in accordance with another preferable embodiment of the present invention;

FIG. 11 depicts operations involved in a facilitator agent response to a service request in accordance with a preferred embodiment of the present invention;

FIG. 12 depicts an Open Agent Architecture™ based system of agents implementing a unified messaging application in accordance with a preferred embodiment of the present invention;

FIG. 13 depicts a map oriented graphical user interface display as might be displayed by a multi-modal map application in accordance with a preferred embodiment of the present invention;

FIG. 14 depicts a peer to peer multiple facilitator based agent system supporting distributed agents in accordance with a preferred embodiment of the present invention;

FIG. 15 depicts a multiple facilitator agent system supporting at least a limited form of a hierarchy of facilitators in accordance with a preferred embodiment of the present invention; and

FIG. 16 depicts a replicated facilitator architecture in accordance with one embodiment of the present invention.

6

BRIEF DESCRIPTION OF THE APPENDICES

The Appendices disclosed in parent application 09/225,198, now U.S. Pat. No. 6,851,115 provide source code for an embodiment of the present invention written in the PROLOG programming language.

APPENDIX A: Source code file named compound.pl.

APPENDIX B: Source code file named fac.pl.

APPENDIX C: Source code file named libcom_tcp.pl.

APPENDIX D: Source code file named liboaa.pl.

APPENDIX E: Source code file named translations.pl.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 3 illustrates a distributed agent system 300 in accordance with one embodiment of the present invention. The agent system 300 includes a facilitator agent 310 and a plurality of agents 320. The illustration of FIG. 3 provides a high level view of one simple system structure contemplated by the present invention. The facilitator agent 310 is in essence the "parent" facilitator for its "children" agents 320. The agents 320 forward service requests to the facilitator agent 310. The facilitator agent 310 interprets these requests, organizing a set of goals which are then delegated to appropriate agents for task completion.

The system 300 of FIG. 3 can be expanded upon and modified in a variety of ways consistent with the present invention. For example, the agent system 300 can be distributed across a computer network such as that illustrated in FIG. 1. The facilitator agent 310 may itself have its functionality distributed across several different computing platforms. The agents 320 may engage in interagent communication (also called peer to peer communications). Several different systems 300 may be coupled together for enhanced performance. These and a variety of other structural configurations are described below in greater detail.

FIG. 4 presents the structure typical of a small system 400 in one embodiment of the present invention, showing user interface agents 408, several application agents 404 and meta-agents 406, the system 400 organized as a community of peers by their common relationship to a facilitator agent 402. As will be appreciated, FIG. 4 places more structure upon the system 400 than shown in FIG. 3, but both are valid representations of structures of the present invention. The facilitator 402 is a specialized server agent that is responsible for coordinating agent communications and cooperative problem-solving. The facilitator 402 may also provide a global data store for its client agents, allowing them to adopt a blackboard style of interaction. Note that certain advantages are found in utilizing two or more facilitator agents within the system 400. For example, larger systems can be assembled from multiple facilitator/client groups, each having the sort of structure shown in FIG. 4. All agents that are not facilitators are referred to herein generically as client agents—so called because each acts (in some respects) as a client of some facilitator, which provides communication and other essential services for the client.

The variety of possible client agents is essentially unlimited. Some typical categories of client agents would include application agents 404, meta-agents 406, and user interface agents 408, as depicted in FIG. 4. Application agents 404 denote specialists that provide a collection of services of a particular sort. These services could be domain-independent technologies (such as speech recognition, natural language processing 410, email, and some forms of data retrieval and data mining) or user-specific or domain-specific (such as a

US 7,069,560 B1

7

travel planning and reservations agent). Application agents may be based on legacy applications or libraries, in which case the agent may be little more than a wrapper that calls a pre-existing API 412, for example. Meta-agents 406 are agents whose role is to assist the facilitator agent 402 in coordinating the activities of other agents. While the facilitator 402 possesses domain-independent coordination strategies, meta-agents 406 can augment these by using domain- and application-specific knowledge or reasoning (including but not limited to rules, learning algorithms and planning).

With further reference to FIG. 4, user interface agents 408 can play an extremely important and interesting role in certain embodiments of the present invention. By way of explanation, in some systems, a user interface agent can be implemented as a collection of "micro-agents", each monitoring a different input modality (point-and-click, handwriting, pen gestures, speech), and collaborating to produce the best interpretation of the current inputs. These micro-agents are depicted in FIG. 4, for example, as Modality Agents 414. While describing such subcategories of client agents is useful for purposes of illustration and understanding, they need not be formally distinguished within the system in preferred implementations of the present invention.

The operation of one preferred embodiment of the present invention will be discussed in greater detail below, but may be briefly outlined as follows. When invoked, a client agent makes a connection to a facilitator, which is known as its parent facilitator. These connections are depicted as a double headed arrow between the client agent and the facilitator agent in FIGS. 3 and 4, for example. Upon connection, an agent registers with its parent facilitator a specification of the capabilities and services it can provide. For example, a natural language agent may register the characteristics of its available natural language vocabulary. (For more details regarding client agent connections, see the discussion of FIG. 8 below.) Later during task completion, when a facilitator determines that the registered services 416 of one of its client agents will help satisfy a goal, the facilitator sends that client a request expressed in the Interagent Communication Language (ICL) 418. (See FIG. 11 below for a more detailed discussion of the facilitator operations involved.) The agent parses this request, processes it, and returns answers or status reports to the facilitator. In processing a request, the client agent can make use of a variety of infrastructure capabilities provided in the preferred embodiment. For example, the client agent can use ICL 418 to request services of other agents, set triggers, and read or write shared data on the facilitator or other client agents that maintain shared data. (See the discussion of FIGS. 9-11 below for a more detailed discussion of request processing.)

The functionality of each client agent are made available to the agent community through registration of the client agent's capabilities with a facilitator 402. A software "wrapper" essentially surrounds the underlying application program performing the services offered by each client. The common infrastructure for constructing agents is preferably supplied by an agent library. The agent library is preferably accessible in the runtime environment of several different programming languages. The agent library preferably minimizes the effort required to construct a new system and maximizes the ease with which legacy systems can be "wrapped" and made compatible with the agent-based architecture of the present invention.

By way of further illustration, a representative application is now briefly presented with reference to FIGS. 5 and 6. In the Automated Office system depicted in FIG. 5, a mobile user with a telephone and a laptop computer can access and

8

task commercial applications such as calendars, databases, and email systems running back at the office. A user interface (UI) agent 408, shown in FIG. 6, runs on the user's local laptop and is responsible for accepting user input, sending requests to the facilitator 402 for delegation to appropriate agents, and displaying the results of the distributed computation. The user may interact directly with a specific remote application by clicking on active areas in the interface, calling up a form or window for that application, and making queries with standard interface dialog mechanisms. Conversely, a user may express a task to be executed by using typed, handwritten, or spoken (over the telephone) English sentences, without explicitly specifying which agent or agents should perform the task.

For instance, if the question "What is my schedule?" is written 420 in the user interface 408, this request will be sent 422 by the UI 408 to the facilitator 402, which in turn will ask 424 a natural language (NL) agent 426 to translate the query into ICL 18. To accomplish this task, the NL agent 426 may itself need to make requests of the agent community to resolve unknown words such as "me" 428 (the UI agent 408 can respond 430 with the name of the current user) or "schedule" 432 (the calendar agent 434 defines this word 436). The resulting ICL expression is then routed by the facilitator 402 to appropriate agents (in this case, the calendar agent 434) to execute the request. Results are sent back 438 to the UI agent 408 for display.

The spoken request "When mail arrives for me about security, notify me immediately." produces a slightly more complex example involving communication among all agents in the system. After translation into ICL as described above, the facilitator installs a trigger 440 on the mail agent 442 to look for new messages about security. When one such message does arrive in its mail pool, the trigger fires, and the facilitator matches the action part of the trigger to capabilities published by the notification agent 446. The notification agent 446 is a meta-agent, as it makes use of rules concerning the optimal use of different output modalities (email, fax, speech generation over the telephone) plus information about an individual user's preferences 448 to determine the best way of relaying a message through available media transfer application agents. After some competitive parallelism to locate the user (the calendar agent 434 and database agent 450 may have different guesses as to where to find the user) and some cooperative parallelism to produce required information (telephone number of location, user password, and an audio file containing a text-to-speech representation of the email message), a telephone agent 452 calls the user, verifying its identity through touchtones, and then play the message.

The above example illustrates a number of inventive features. As new agents connect to the facilitator, registering capability specifications and natural language vocabulary, what the user can say and do dynamically changes; in other words, the ICL is dynamically expandable. For example, adding a calendar agent to the system in the previous example and registering its capabilities enables users to ask natural language questions about their "schedule" without any need to revise code for the facilitator, the natural language agents, or any other client agents. In addition, the interpretation and execution of a task is a distributed process, with no single agent defining the set of possible inputs to the system. Further, a single request can produce cooperation and flexible communication among many agents, written in different programming languages and spread across multiple machines.

US 7,069,560 B1

9

Design Philosophy and Considerations

One preferred embodiment provides an integration mechanism for heterogeneous applications in a distributed infrastructure, incorporating some of the dynamism and extensibility of blackboard approaches, the efficiency associated with mobile objects, plus the rich and complex interactions of communicating agents. Design goals for preferred embodiments of the present invention may be categorized under the general headings of interoperation and cooperation, user interfaces, and software engineering. These design goals are not absolute requirements, nor will they necessarily be satisfied by all embodiments of the present invention, but rather simply reflect the inventor's currently preferred design philosophy.

Versatile Mechanisms of Interoperation and Cooperation

Interoperation refers to the ability of distributed software components—agents—to communicate meaningfully. While every system-building framework must provide mechanisms of interoperation at some level of granularity, agent-based frameworks face important new challenges in this area. This is true primarily because autonomy, the hallmark of individual agents, necessitates greater flexibility in interactions within communities of agents. Coordination refers to the mechanisms by which a community of agents is able to work together productively on some task. In these areas, the goals for our framework are to provide flexibility in assembling communities of autonomous service providers, provide flexibility in structuring cooperative interactions, impose the right amount of structure, as well as include legacy and “owned-elsewhere” applications.

Provide flexibility in assembling communities of autonomous service providers—both at development time and at runtime. Agents that conform to the linguistic and ontological requirements for effective communication should be able to participate in an agent community, in various combinations, with minimal or near minimal prerequisite knowledge of the characteristics of the other players. Agents with duplicate and overlapping capabilities should be able to coexist within the same community, with the system making optimal or near optimal use of the redundancy.

Provide flexibility in structuring cooperative interactions among the members of a community of agents. A framework preferably provides an economical mechanism for setting up a variety of interaction patterns among agents, without requiring an inordinate amount of complexity or infrastructure within the individual agents. The provision of a service should be independent or minimally dependent upon a particular configuration of agents.

Impose the right amount of structure on individual agents. Different approaches to the construction of multi-agent systems impose different requirements on the individual agents. For example, because KQML is neutral as to the content of messages, it imposes minimal structural requirements on individual agents. On the other hand, the BDI paradigm tends to impose much more demanding requirements, by making assumptions about the nature of the programming elements that are meaningful to individual agents. Preferred embodiments of the present invention should fall somewhere between the two, providing a rich set of interoperation and coordination capabilities, without precluding any of the software engineering goals defined below.

Include legacy and “owned-elsewhere” applications. Whereas legacy usually implies reuse of an established system fully controlled by the agent-based system developer, owned-elsewhere refers to applications to which the developer has partial access, but no control. Examples of

10

owned-elsewhere applications include data sources and services available on the World Wide Web, via simple form-based interfaces, and applications used cooperatively within a virtual enterprise, which remain the properties of separate corporate entities. Both classes of application must preferably be able to interoperate, more or less as full-fledged members of the agent community, without requiring an overwhelming integration effort.

Human-Oriented User Interfaces

Systems composed of multiple distributed components, and possibly dynamic configurations of components, require the crafting of intuitive user interfaces to provide conceptually natural interaction mechanisms, treat users as privileged members of the agent community and support collaboration.

Provide conceptually natural interaction mechanisms with multiple distributed components. When there are numerous disparate agents, and/or complex tasks implemented by the system, the user should be able to express requests without having detailed knowledge of the individual agents. With speech recognition, handwriting recognition, and natural language technologies becoming more mature, agent architectures should preferably support these forms of input playing increased roles in the tasking of agent communities.

Preferably treat users as privileged members of the agent community by providing an appropriate level of task specification within software agents, and reusable translation mechanisms between this level and the level of human requests, supporting constructs that seamlessly incorporate interactions between both human-interface and software types of agents.

Preferably support collaboration (simultaneous work over shared data and processing resources) between users and agents. Realistic software engineering requirements System-building frameworks should preferably address the practical concerns of real-world applications by the specification of requirements which preferably include: Minimize the effort required to create new agents, and to wrap existing applications. Encourage reuse, both of domain-independent and domain-specific components. The concept of agent orientation, like that of object orientation, provides a natural conceptual framework for reuse, so long as mechanisms for encapsulation and interaction are structured appropriately. Support lightweight, mobile platforms. Such platforms should be able to serve as hosts for agents, without requiring the installation of a massive environment. It should also be possible to construct individual agents that are relatively small and modest in their processing requirements. Minimize platform and language barriers. Creation of new agents, as well as wrapping of existing applications, should not require the adoption of a new language or environment.

Mechanisms of Cooperation

Cooperation among agents in accordance with the present invention is preferably achieved via messages expressed in a common language, ICL. Cooperation among agent is further preferably structured around a three-part approach: providers of services register capabilities specifications with a facilitator, requesters of services construct goals and relay them to a facilitator, and facilitators coordinate the efforts of the appropriate service providers in satisfying these goals. The Interagent Communication Language (ICL) Interagent Communication Language (“ICL”) 418 refers to an interface, communication, and task coordination language preferably shared by all agents, regardless of what platform they run on or what computer language they are programmed in. ICL may be used by an agent to task itself or some subset

US 7,069,560 B1

11

of the agent community. Preferably, ICL allows agents to specify explicit control parameters while simultaneously supporting expression of goals in an underspecified, loosely constrained manner. In a further preferred embodiment, agents employ ICL to perform queries, execute actions, exchange information, set triggers, and manipulate data in the agent community.

In a further preferred embodiment, a program element expressed in ICL is the event. The activities of every agent, as well as communications between agents, are preferably structured around the transmission and handling of events. In communications, events preferably serve as messages between agents; in regulating the activities of individual agents, they may preferably be thought of as goals to be satisfied. Each event preferably has a type, a set of parameters, and content. For example, the agent library procedure `oaa_Solve` can be used by an agent to request services of other agents. A call to `oaa_Solve`, within the code of agent A, results in an event having the form

```
ev_post_solve(Goal, Params)
```

going from A to the facilitator, where `ev_post_solve` is the type, `Goal` is the content, and `Params` is a list of parameters. The allowable content and parameters preferably vary according to the type of the event.

The ICL preferably includes a layer of conversational protocol and a content layer. The conversational layer of ICL is defined by the event types, together with the parameter lists associated with certain of these event types. The content layer consists of the specific goals, triggers, and data elements that may be embedded within various events.

The ICL conversational protocol is preferably specified using an orthogonal, parameterized approach, where the conversational aspects of each element of an interagent conversation are represented by a selection of an event type and a selection of values from at least one orthogonal set of parameters. This approach offers greater expressiveness than an approach based solely on a fixed selection of speech acts, such as embodied in KQML. For example, in KQML, a request to satisfy a query can employ either of the performatives `ask_all` or `ask_one`. In ICL, on the other hand, this type of request preferably is expressed by the event type `ev_post_solve`, together with the `solution_limit(N)` parameter—where `N` can be any positive integer. (A request for all solutions is indicated by the omission of the `solution_limit` parameter.) The request can also be accompanied by other parameters, which combine to further refine its semantics. In KQML, then, this example forces one to choose between two possible conversational options, neither of which may be precisely what is desired. In either case, the performative chosen is a single value that must capture the entire conversational characterization of the communication. This requirement raises a difficult challenge for the language designer, to select a set of performatives that provides the desired functionality without becoming unmanageably large. Consequently, the debate over the right set of performatives has consumed much discussion within the KQML community.

The content layer of the ICL preferably supports unification and other features found in logic programming language environments such as PROLOG. In some embodiments, the content layer of the ICL is simply an extension of at least one programming language. For example, the Applicants have found that PROLOG is suitable for implementing and extending into the content layer of the ICL. The agent libraries preferably provide support for constructing, pars-

12

ing, and manipulating ICL expressions. It is possible to embed content expressed in other languages within an ICL event. However, expressing content in ICL simplifies the facilitator's access to the content, as well as the conversational layer, in delegating requests. This gives the facilitator more information about the nature of a request and helps the facilitator decompose compound requests and delegate the sub-requests.

Further, ICL expressions preferably include, in addition to events, at least one of the following: capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements. A further preferred embodiment of the present invention incorporates ICL expressions including at least all of the following: events, capabilities declarations, requests for services, responses to requests, trigger specifications, and shared data elements.

Providing Services: Specifying "Solvables"

In a preferred embodiment of the present invention, every participating agent defines and publishes a set of capability declarations, expressed in ICL, describing the services that it provides. These declarations establish a high-level interface to the agent. This interface is used by a facilitator in communicating with the agent, and, most important, in delegating service requests (or parts of requests) to the agent. Partly due to the use of PROLOG as a preferred basis for ICL, these capability declarations are referred as solvables. The agent library preferably provides a set of procedures allowing an agent to add, remove, and modify its solvables, which it may preferably do at any time after connecting to its facilitator.

There are preferably at least two major types of solvables: procedure solvables and data solvables. Intuitively, a procedure solvable performs a test or action, whereas a data solvable provides access to a collection of data. For example, in creating an agent for a mail system, procedure solvables might be defined for sending a message to a person, testing whether a message about a particular subject has arrived in the mail queue, or displaying a particular message onscreen. For a database wrapper agent, one might define a distinct data solvable corresponding to each of the relations present in the database. Often, a data solvable is used to provide a shared data store, which may be not only queried, but also updated, by various agents having the required permissions.

There are several primary technical differences between these two types of solvables. First, each procedure solvable must have a handler declared and defined for it, whereas this is preferably not necessary for a data solvable. The handling of requests for a data solvable is preferably provided transparently by the agent library. Second, data solvables are preferably associated with a dynamic collection of facts (or clauses), which may be further preferably modified at runtime, both by the agent providing the solvable, and by other agents (provided they have the required permissions). Third, special features, available for use with data solvables, preferably facilitate maintaining the associated facts. In spite of these differences, it should be noted that the mechanism of use by which an agent requests a service is the same for the two types of solvables.

In one embodiment, a request for one of an agent's services normally arrives in the form of an event from the agent's facilitator. The appropriate handler then deals with this event. The handler may be coded in whatever fashion is most appropriate, depending on the nature of the task, and the availability of task-specific libraries or legacy code, if any. The only hard requirement is that the handler return an

US 7,069,560 B1

13

appropriate response to the request, expressed in ICL. Depending on the nature of the request, this response could be an indication of success or failure, or a list of solutions (when the request is a data query).

A solvable preferably has three parts: a goal, a list of parameters, and a list of permissions, which are declared using the format:

```
solvable(Goal, Parameters, Permissions)
```

The goal of a solvable, which syntactically takes the preferable form of an ICL structure, is a logical representation of the service provided by the solvable. (An ICL structure consists of a functor with 0 or more arguments. For example, in the structure $a(b,c)$, 'a' is the functor, and 'b' and 'c' the arguments.) As with a PROLOG structure, the goal's arguments themselves may preferably be structures.

Various options can be included in the parameter list, to refine the semantics associated with the solvable. The type parameter is preferably used to say whether the solvable is data or procedure. When the type is procedure, another parameter may be used to indicate the handler to be associated with the solvable. Some of the parameters appropriate for a data solvable are mentioned elsewhere in this application. In either case (procedure or data solvable), the private parameter may be preferably used to restrict the use of a solvable to the declaring agent when the agent intends the solvable to be solely for its internal use but wishes to take advantage of the mechanisms in accordance with the present invention to access it, or when the agent wants the solvable to be available to outside agents only at selected times. In support of the latter case, it is preferable for the agent to change the status of a solvable from private to non-private at any time.

The permissions of a solvable provide mechanisms by which an agent may preferably control access to its services allowing the agent to restrict calling and writing of a solvable to itself and/or other selected agents. (Calling means requesting the service encapsulated by a solvable, whereas writing means modifying the collection of facts associated with a data solvable.) The default permission for every solvable in a further preferred embodiment of the present invention is to be callable by anyone, and for data solvables to be writable by anyone. A solvable's permissions can preferably be changed at any time, by the agent providing the solvable.

For example, the solvables of a simple email agent might include:

```
solvable(send_message(email,+ToPerson,+Params),
  [type(procedure), callback(send_mail)],
[])
solvable(last_message(email,-MessageId),
  [type(data), single_(true)],
  [write(true)],
solvable(get_message(email,+MessageId,-Msg),
  [type(procedure), callback(get_mail)],
[])
```

The symbols '+' and '-', indicating input and output arguments, are at present used only for purposes of documentation. Most parameters and permissions have default values, and specifications of default values may be omitted from the parameters and permissions lists.

Defining an agent's capabilities in terms of solvable declarations effectively creates a vocabulary with which other agents can communicate with the new agent. Ensuring that agents will speak the same language and share a common, unambiguous semantics of the vocabulary involves ontology. Agent development tools and services (automatic translations of solvables by the facilitator) help

14

address this issue; additionally, a preferred embodiment of the present invention will typically rely on vocabulary from either formally engineered ontologies for specific domains or from ontologies constructed during the incremental development of a body of agents for several applications or from both specific domain ontologies and incrementally developed ontologies. Several example tools and services are described in Cheyer et al.'s paper entitled "Development Tools for the Open Agent Architecture," as presented at the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 96), London, April 1996.

Although the present invention imposes no hard restrictions on the form of solvable declarations, two common usage conventions illustrate some of the utility associated with solvables.

Classes of services are often preferably tagged by a particular type. For instance, in the example above, the "last_message" and "get_message" solvables are specialized for email, not by modifying the names of the services, but rather by the use of the 'email' parameter, which serves during the execution of an ICL request to select (or not) a specific type of message.

Actions are generally written using an imperative verb as the functor of the solvable in a preferred embodiment of the present invention, the direct object (or item class) as the first argument of the predicate, required arguments following, and then an extensible parameter list as the last argument. The parameter list can hold optional information usable by the function. The ICL expression generated by a natural language parser often makes use of this parameter list to store prepositional phrases and adjectives.

As an illustration of the above two points, "Send mail to Bob about lunch" will be translated into an ICL request `send_message(email, 'Bob Jones', [subject(lunch)])`, whereas "Remind Bob about lunch" would leave the transport unspecified (`send_message(KIND, 'Bob Jones', [subject(lunch)])`), enabling all available message transfer agents (e.g., fax, phone, mail, pager) to compete for the opportunity to carry out the request.

Requesting Services

An agent preferably requests services of the community of agent by delegating tasks or goals to its facilitator. Each request preferably contains calls to one or more agent solvables, and optionally specifies parameters containing advice to help the facilitator determine how to execute the task. Calling a solvable preferably does not require that the agent specify (or even know of) a particular agent or agents to handle the call. While it is possible to specify one or more agents using an address parameter (and there are situations in which this is desirable), in general it is advantageous to leave this delegation to the facilitator. This greatly reduces the hard-coded component dependencies often found in other distributed frameworks. The agent libraries of a preferred embodiment of the present invention provide an agent with a single, unified point of entry for requesting services of other agents: the library procedure `oaa_Solve`. In the style of logic programming, `oaa_Solve` may preferably be used both to retrieve data and to initiate actions, so that calling a data solvable looks the same as calling a procedure solvable.

Complex Goal Expressions

A powerful feature provided by preferred embodiments of the present invention is the ability of a client agent (or a user) to submit compound goals of an arbitrarily complex nature to a facilitator. A compound goal is a single goal expression that specifies multiple sub-goals to be performed. In speaking of a "complex goal expression" we mean that a single

US 7,069,560 B1

15

goal expression that expresses multiple sub-goals can potentially include more than one type of logical connector (e.g., AND, OR, NOT), and/or more than one level of logical nesting (e.g., use of parentheses), or the substantive equivalent. By way of further clarification, we note that when speaking of an "arbitrarily complex goal expression" we mean that goals are expressed in a language or syntax that allows expression of such complex goals when appropriate or when desired, not that every goal is itself necessarily complex.

It is contemplated that this ability is provided through an interagent communication language having the necessary syntax and semantics. In one example, the goals may take the form of compound goal expressions composed using operators similar to those employed by PROLOG, that is, the comma for conjunction, the semicolon for disjunction, the arrow for conditional execution, etc. The present invention also contemplates significant extensions to PROLOG syntax and semantics. For example, one embodiment incorporates a "parallel disjunction" operator indicating that the disjuncts are to be executed by different agents concurrently. A further embodiment supports the specification of whether a given sub-goal is to be executed breadth-first or depth-first.

A further embodiment supports each sub-goal of a compound goal optionally having an address and/or a set of parameters attached to it. Thus, each sub-goal takes the form

Address: Goal::Parameters

where both Address and Parameters are optional.

An address, if present, preferably specifies one or more agents to handle the given goal, and may employ several different types of referring expression: unique names, symbolic names, and shorthand names. Every agent has preferably a unique name, assigned by its facilitator, which relies upon network addressing schemes to ensure its global uniqueness. Preferably, agents also have self-selected symbolic names (for example, "mail"), which are not guaranteed to be unique. When an address includes a symbolic name, the facilitator preferably takes this to mean that all agents having that name should be called upon. Shorthand names include 'self' and 'parent' (which refers to the agent's facilitator). The address associated with a goal or sub-goal is preferably always optional. When an address is not present, it is the facilitator's job to supply an appropriate address.

The distributed execution of compound goals becomes particularly powerful when used in conjunction with natural language or speech-enabled interfaces, as the query itself may specify how functionality from distinct agents will be combined. As a simple example, the spoken utterance "Fax it to Bill Smith's manager." can be translated into the following compound ICL request:

```
oaa_Solve((manager('Bill Smith', M), fax(it,M,[ ])),
[strategy(action)])
```

Note that in this ICL request there are two sub-goals, "manager('Bill Smith',M)" and "fax(it,M,[])," and a single global parameter "strategy(action)." According to the present invention, the facilitator is capable of mapping global parameters in order to apply the constraints or advice across the separate sub-goals in a meaningful way. In this instance, the global parameter strategy(action) implies a parallel constraint upon the first sub-goal; i.e., when there are multiple agents that can respond to the manager sub-goal, each agent should receive a request for service. In contrast, for the second sub-goal, parallelism should not be

16

inferred from the global parameter strategy(action) because such an inference would possibly result in the transmission of duplicate facsimiles.

Refining Service Requests

In a preferred embodiment of the present invention, parameters associated with a goal (or sub-goal) can draw on useful features to refine the request's meaning. For example, it is frequently preferred to be able to specify whether or not solutions are to be returned synchronously; this is done using the reply parameter, which can take any of the values synchronous, asynchronous, or none. As another example, when the goal is a non-compound query of a data solvable, the cache parameter may preferably be used to request local caching of the facts associated with that solvable. Many of the remaining parameters fall into two categories: feedback and advice.

Feedback parameters allow a service requester to receive information from the facilitator about how a goal was handled. This feedback can include such things as the identities of the agents involved in satisfying the goal, and the amount of time expended in the satisfaction of the goal.

Advice parameters preferably give constraints or guidance to the facilitator in completing and interpreting the goal. For example, a solution_limit parameter preferably allows the requester to say how many solutions it is interested in; the facilitator and/or service providers are free to use this information in optimizing their efforts. Similarly, a time_limit is preferably used to say how long the requester is willing to wait for solutions to its request, and, in a multiple facilitator system, a level_limit may preferably be used to say how remote the facilitators may be that are consulted in the search for solutions. A priority parameter is preferably used to indicate that a request is more urgent than previous requests that have not yet been satisfied. Other preferred advice parameters include but are not limited to parameters used to tell the facilitator whether parallel satisfaction of the parts of a goal is appropriate, how to combine and filter results arriving from multiple solver agents, and whether the requester itself may be considered a candidate solver of the sub-goals of a request.

Advice parameters preferably provide an extensible set of low-level, orthogonal parameters capable of combining with the ICL goal language to fully express how information should flow among participants. In certain preferred embodiments of the present invention, multiple parameters can be grouped together and given a group name. The resulting high-level advice parameters can preferably be used to express concepts analogous to KQML's performatives, as well as define classifications of problem types. For instance, KQML's "ask_all" and "ask_one" performatives would be represented as combinations of values given to the parameters reply, parallel ok, and solution_limit. As an example of a higher-level problem type, the strategy "math_problem" might preferably send the query to all appropriate math solvers in parallel, collect their responses, and signal a conflict if different answers are returned. The strategy "essay_question" might preferably send the request to all appropriate participants, and signal a problem (i.e., cheating) if any of the returned answers are identical.

Facilitation

In a preferred embodiment of the present invention, when a facilitator receives a compound goal, its job is to construct a goal satisfaction plan and oversee its satisfaction in an optimal or near optimal manner that is consistent with the specified advice. The facilitator of the present invention maintains a knowledge base that records the capabilities of

US 7,069,560 B1

17

a collection of agents, and uses that knowledge to assist requesters and providers of services in making contact.

FIG. 7 schematically shows data structures 700 internal to a facilitator in accordance with one embodiment of the present invention. Consider the function of a Agent Registry 702 in the present invention. Each registered agent may be seen as associated with a collection of fields found within its parent facilitator such as shown in the figure. Each registered agent may optionally possess a Symbolic Name which would be entered into field 704. As mentioned elsewhere, Symbolic Names need not be unique to each instance of an agent. Note that an agent may in certain preferred embodiments of the present invention possess more than one Symbolic Name. Such Symbolic Names would each be found through their associations in the Agent Registry entries. Each agent, when registered, must possess a Unique Address, which is entered into the Unique Address field 706.

With further reference to FIG. 7, each registered agent may be optionally associated with one or more capabilities, which have associated Capability Declaration fields 708 in the parent facilitator Agent Registry 702. These capabilities may define not just functionality, but may further provide a utility parameter indicating, in some manner (e.g., speed, accuracy, etc), how effective the agent is at providing the declared capability. Each registered agent may be optionally associated with one or more data components, which have associated Data Declaration fields 710 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more triggers, which preferably could be referenced through their associated Trigger Declaration fields 712 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more tasks, which preferably could be referenced through their associated Task Declaration fields 714 in the parent facilitator Agent Registry 702. Each registered agent may be optionally associated with one or more Process Characteristics, which preferably could be referenced through their associated Process Characteristics Declaration fields 716 in the parent facilitator Agent Registry 702. Note that these characteristics in certain preferred embodiments of the present invention may include one or more of the following: Machine Type (specifying what type of computer may run the agent), Language (both computer and human interface).

A facilitator agent in certain preferred embodiments of the present invention further includes a Global Persistent Database 720. The database 720 is composed of data elements which do not rely upon the invocation or instantiation of client agents for those data elements to persist. Examples of data elements which might be present in such a database include but are not limited to the network address of the facilitator agent's server, facilitator agent's server accessible network port list, firewalls, user lists, and security options regarding the access of server resources accessible to the facilitator agent.

A simplified walk through of operations involved in creating a client agent, a client agent initiating a service request, a client agent responding to a service request and a facilitator agent responding to a service request are including hereafter by way of illustrating the use of such a system. These figures and their accompanying discussion are provided by way of illustration of one preferred embodiment of the present invention and are not intended to limit the scope of the present invention.

FIG. 8 depicts operations involved in instantiating a client agent with its parent facilitator in accordance with a preferred embodiment of the present invention. The operations

18

begin with starting the Agent Registration in a step 800. In a next step 802, the Installer, such as a 955 client or facilitator agent, invokes a new client agent. It will be appreciated that any computer entity is capable of invoking a new agent.

The system then instantiates the new client agent in a step 804. This operation may involve resource allocations somewhere in the network on a local computer system for the client agent, which will often include 960 memory as well as placement of references to the newly instantiated client agent in internal system lists of agents within that local computing system. Once instantiated, the new client and its parent facilitator establish a communications link in a step 806. In certain preferred embodiments, this communications link involves selection of one or more physical transport mechanisms for this communication. Once established, the client agent transmits its profile to the parent facilitator in a step 808. When received, the parent facilitator registers the client agent in a step 810. Then, at a step 812, a client agent has been instantiated in accordance with one preferred embodiment of the present invention.

FIG. 9 depicts operations involved in a client agent initiating a service request and receiving the response to that service request in accordance with a preferred embodiment of the present invention. The method of FIG. 9 begins in a step 900, wherein any initialization or other such procedures may be performed. Then, in a step 902, the client 975 agent determines a goal to be achieved (or solved). This goal is then translated in a step 904 into ICL, if it is not already formulated in it. The goal, now stated in ICL, is then transmitted to the client agent's parent facilitator in a step 906. The parent facilitator responds to this service request and at a later time, the client agent receives the results of the 980 request in a step 908, operations of FIG. 9 being complete in a done step 910.

FIG. 10 depicts operations involved in a client agent responding to a service request in accordance with a preferred embodiment of the present invention. Once started in a step 1000, the client agent receives the service request in a step 1002. In a next step 1004, the client agent parses the received request from ICL. The client agent then determines if the service is available in a step 1006. If it is not, the client agent returns a status report to that effect in a step 1008. If the service is available, control is passed to a step 1010 where the client performs the requested service. Note that in completing step 1010 the client may form complex goal expressions, requesting results for these solvables from the facilitator agent. For example, a fax agent might fax a document to a certain person only after requesting and receiving a fax number for that person. Subsequently, the client agent either returns the results of the service and/or a status report in a step 1012. The operations of FIG. 10 are complete in a done step 1014.

FIG. 11 depicts operations involved in a facilitator agent response to a service request in accordance with a preferred embodiment of the present invention. The start of such operations in step 1100 leads to the reception of a goal request in a step 1102 by the facilitator. This request is then parsed and interpreted by the facilitator in a step 1104. The facilitator then proceeds to construct a goal satisfaction plan in a next step 1106. In steps 1108 and 1110, respectively, the facilitator determines the required sub-goals and then selects agents suitable for performing the required sub-goals. The facilitator then transmits the sub-goal requests to the selected agents in a step 1112 and receives the results of these transmitted requests in a step 1114. It should be noted that the actual implementation of steps 1112 and 1114 are

US 7,069,560 B1

19

dependent upon the specific goal satisfaction plan. For instance, certain sub-goals may be sent to separate agents in parallel, while transmission of other sub-goals may be postponed until receipt of particular answers. Further, certain requests may generate multiple responses that generate additional sub-goals. Once the responses have been received, the facilitator determines whether the original requested goal has been completed in a step 1118. If the original requested goal has not been completed, the facilitator recursively repeats the operations 1106 through 1116. Once the original requested goal is completed, the facilitator returns the results to the requesting agent 1118 and the operations are done at 1120.

A further preferred embodiment of the present invention incorporates transparent delegation, which means that a requesting agent can generate a request, and a facilitator can manage the satisfaction of that request, without the requester needing to have any knowledge of the identities or locations of the satisfying agents. In some cases, such as when the request is a data query, the requesting agent may also be oblivious to the number of agents involved in satisfying a request. Transparent delegation is possible because agents' capabilities (solvable) are treated as an abstract description of a service, rather than as an entry point into a library or body of code.

A further preferred embodiment of the present invention incorporates facilitator handling of compound goals, preferably involving three types of processing: delegation, optimization and interpretation.

Delegation processing preferably supports facilitator determination of which specific agents will execute a compound goal and how such a compound goal's sub-goals will be combined and the sub-goal results routed. Delegation involves selective application of global and local constraint and advice parameters onto the specific sub-goals. Delegation results in a goal that is unambiguous as to its meaning and as to the agents that will participate in satisfying it.

Optimization processing of the completed goal preferably includes the facilitator using sub-goal parallelization where appropriate. Optimization results in a goal whose interpretation will require as few exchanges as possible, between the facilitator and the satisfying agents, and can exploit parallel efforts of the satisfying agents, wherever this does not affect the goal's meaning.

Interpretation processing of the optimized goal. Completing the addressing of a goal involves the selection of one or more agents to handle each of its sub-goals (that is, each sub-goal for which this selection has not been specified by the requester). In doing this, the facilitator uses its knowledge of the capabilities of its client agents (and possibly of other facilitators, in a multi-facilitator system). It may also use strategies or advice specified by the requester, as explained below. The interpretation of a goal involves the coordination of requests to the satisfying agents, and assembling their responses into a coherent whole, for return to the requester.

A further preferred embodiment of present invention extends facilitation so the facilitator can employ strategies and advice given by the requesting agent, resulting in a variety of interaction patterns that may be instantiated in the satisfaction of a request.

A further preferred embodiment of present invention handles the distribution of both data update requests and requests for installation of triggers, preferably using some of the same strategies that are employed in the delegation of service requests.

20

Note that the reliance on facilitation is not absolute; that is, there is no hard requirement that requests and services be matched up by the facilitator, or that interagent communications go through the facilitator. There is preferably support in the agent library for explicit addressing of requests. However, a preferred embodiment of the present invention encourages employment the paradigm of agent communities, minimizing their development effort, by taking advantage of the facilitator's provision of transparent delegation and handling of compound goals.

A facilitator is preferably viewed as a coordinator, not a controller, of cooperative task completion. A facilitator preferably never initiates an activity. A facilitator preferably responds to requests to manage the satisfaction of some goal, the update of some data repository, or the installation of a trigger by the appropriate agent or agents. All agents can preferably take advantage of the facilitator's expertise in delegation, and its up-to-date knowledge about the current membership of a dynamic community. The facilitator's coordination services often allows the developer to lessen the complexity of individual agents, resulting in a more manageable software development process, and enabling the creation of lightweight agents.

25 Maintaining Data Repositories

The agent library supports the creation, maintenance, and use of databases, in the form of data solvable. Creation of a data solvable requires only that it be declared. Querying a data solvable, as with access to any solvable, is done using oaa_solve.

A data solvable is conceptually similar to a relation in a relational database. The facts associated with each solvable are maintained by the agent library, which also handles incoming messages containing queries of data solvables. The default behavior of an agent library in managing these facts may preferably be refined, using parameters specified with the solvable's declaration. For example, the parameter single_value preferably indicates that the solvable should only contain a single fact at any given point in time. The parameter unique_values preferably indicates that no duplicate values should be stored.

Other parameters preferably allow data solvables use of the concepts of ownership and persistence. For implementing shared repositories, it is often preferable to maintain a record of which agent created each fact of a data solvable with the creating agent being preferably considered the fact's owner. In many applications, it is preferable to remove an agent's facts when that agent goes offline (for instance, when the agent is no longer participating in the agent community, whether by deliberate termination or by malfunction). When a data solvable is declared to be non-persistent, its facts are automatically maintained in this way, whereas a persistent data solvable preferably retains its facts until they are explicitly removed.

A further preferred embodiment of present invention supports an agent library through procedures by which agents can update (add, remove, and replace) facts belonging to data solvables, either locally or on other agents, given that they have preferably the required permissions. These procedures may preferably be refined using many of the same parameters that apply to service requests. For example, the address parameter preferably specifies one or more particular agents to which the update request applies. In its absence, just as with service requests, the update request preferably goes to all agents providing the relevant data solvable. This default behavior can be used to maintain

US 7,069,560 B1

21

coordinated "mirror" copies of a data set within multiple agents, and can be useful in support of distributed, collaborative activities.

Similarly, the feedback parameters, described in connection with oaa_Solve, are preferably available for use with data maintenance requests.

A further preferred embodiment of present invention supports ability to provide data solvables not just to client agents, but also to facilitator agents. Data solvables can preferably be created, maintained and used by a facilitator. The facilitator preferably can, at the request of a client of the facilitator, create, maintain and share the use of data solvables with all the facilitator's clients. This can be useful with relatively stable collections of agents, where the facilitator's workload is predictable. Using a Blackboard Style of Communication In a further preferred embodiment of present invention, when a data solvable is publicly readable and writable, it acts essentially as a global data repository and can be used cooperatively by a group of agents. In combination with the use of triggers, this allows the agents to organize their efforts around a "blackboard" style of communication.

As an example, the "DCG-NL" agent (one of several existing natural language processing agents), provides natural language processing services for a variety of its peer agents, expects those other agents to record, on the facilitator, the vocabulary to which they are prepared to respond, with an indication of each word's part of speech, and of the logical form (ICL sub-goal) that should result from the use of that word. In a further preferred embodiment of present invention, the NL agent, preferably when it comes online, preferably installs a data solvable for each basic part of speech on its facilitator. For instance, one such solvable would be:

```
solvable(noun(Meaning, Syntax), [ ], [ ])
```

Note that the empty lists for the solvable's permissions and parameters are acceptable here, since the default permissions and parameters provide appropriate functionality.

A further preferred embodiment of present invention incorporating an Office Assistant system as discussed herein or similar to the discussion here supports several agents making use of these or similar services. For instance, the database agent uses the following call, to library procedure oaa_AddData, to post the noun 'boss', and to indicate that the "meaning" of boss is the concept 'manager':

```
oaa_AddData(noun(manager, atom(boss)), [address  
(parent)]) Autonomous Monitoring with Triggers
```

A further preferred embodiment of present invention includes support for triggers, providing a general mechanism for requesting some action be taken when a set of conditions is met. Each agent can preferably install triggers either locally, for itself, or remotely, on its facilitator or peer agents. There are preferably at least four types of triggers: communication, data, task, and time. In addition to a type, each trigger preferably specifies at least a condition and an action, both preferably expressed in ICL. The condition indicates under what circumstances the trigger should fire, and the action indicates what should happen when it fires. In addition, each trigger can be set to fire either an unlimited number of times, or a specified number of times, which can be any positive integer.

Triggers can be used in a variety of ways within preferred embodiments of the present invention. For example, triggers can be used for monitoring external sensors in the execution

22

environment, tracking the progress of complex tasks, or coordinating communications between agents that are essential for the synchronization of related tasks. The installation of a trigger within an agent can be thought of as a representation of that agent's commitment to carry out the specified action, whenever the specified condition holds true.

Communication triggers preferably allow any incoming or outgoing event (message) to be monitored. For instance, a simple communication trigger may say something like: "Whenever a solution to a goal is returned from the facilitator, send the result to the presentation manager to be displayed to the user."

Data triggers preferably monitor the state of a data repository (which can be maintained on a facilitator or a client agent). Data triggers' conditions may be tested upon the addition, removal, or replacement of a fact belonging to a data solvable. An example data trigger is: "When 15 users are simultaneously logged on to a machine, send an alert message to the system administrator."

Task triggers preferably contain conditions that are tested after the processing of each incoming event and whenever a timeout occurs in the event polling. These conditions may specify any goal executable by the local ICL interpreter, and most often are used to test when some solvable becomes satisfiable. Task triggers are useful in checking for task-specific internal conditions. Although in many cases such conditions are captured by solvables, in other cases they may not be. For example, a mail agent might watch for new incoming mail, or an airline database agent may monitor which flights will arrive later than scheduled. An example task trigger is: "When mail arrives for me about security, notify me immediately."

Time triggers preferably monitor time conditions. For instance, an alarm trigger can be set to fire at a single fixed point in time (e.g., "On December 23rd at 3 pm"), or on a recurring basis (e.g., "Every three minutes from now until noon").

Triggers are preferably implemented as data solvables, declared implicitly for every agent. When requesting that a trigger be installed, an agent may use many of the same parameters that apply to service and data maintenance requests.

A further preferred embodiment of present invention incorporates semantic support, in contrast with most programming methodologies, of the agent on which the trigger is installed only having to know how to evaluate the conditional part of the trigger, not the consequence. When the trigger fires, the action is delegated to the facilitator for execution. Whereas many commercial mail programs allow rules of the form "When mail arrives about XXX, [forward it, delete it, archive it]", the possible actions are hard-coded and the user must select from a fixed set.

A further preferred embodiment of present invention, the consequence of a trigger may be any compound goal executable by the dynamic community of agents. Since new agents preferably define both functionality and vocabulary, when an unanticipated agent (for example, a fax agent) joins the community, no modifications to existing code is required for a user to make use of it—"When mail arrives, fax it to Bill Smith."

The Agent Library

In a preferred embodiment of present invention, the agent library provides the infrastructure for constructing an agent-based system. The essential elements of protocol (involving the details of the messages that encapsulate a service request

US 7,069,560 B1

23

and its response) are preferably made transparent to simplify the programming applications. This enables the developer to focus functionality, rather than message construction details and communication details. For example, to request a service of another agent, an agent preferably calls the library procedure `oaa_Solve`. This call results in a message to a facilitator, which will exchange messages with one or more service providers, and then send a message containing the desired results to the requesting agent. These results are returned via one of the arguments of `oaa_Solve`. None of the messages involved in this scenario is explicitly constructed by the agent developer. Note that this describes the synchronous use of `oaa_Solve`.

In another preferred embodiment of present invention, an agent library provides both intraagent and interagent infrastructure; that is, mechanisms supporting the internal structure of individual agents, on the one hand, and mechanisms of cooperative interoperation between agents, on the other. Note that most of the infrastructure cuts across this boundary with many of the same mechanisms supporting both agent internals and agent interactions in an integrated fashion. For example, services provided by an agent preferably can be accessed by that agent through the same procedure (`oaa_Solve`) that it would employ to request a service of another agent (the only difference being in the address parameter accompanying the request). This helps the developer to reuse code and avoid redundant entry points into the same functionality.

Both of the preferred characteristics described above (transparent construction of messages and integration of intraagent with interagent mechanisms) apply to most other library functionality as well, including but not limited to data management and temporal control mechanisms.

Source Code Appendix

Source code for version 2.0 of the OAA software product is included as an appendix hereto, and is incorporated herein by reference. The code includes an agent library, which provides infrastructure for constructing an agent-based system. The library's several families of procedures provide the functionalities discussed above, as well as others that have not been discussed here but that will be sufficiently clear to the interested practitioner. For example, declarations of an agent's solvables, and their registration with a facilitator, are managed using procedures such as `oaa_Declare`, `oaa_Undeclare`, and `oaa_Redeclare`. Updates to data solvables can be accomplished with a family of procedures including `oaa_AddData`, `oaa_RemoveData`, and `oaa_ReplaceData`. Similarly, triggers are maintained using procedures such as `oaa_AddTrigger`, `oaa_RemoveTrigger`, and `oaa_ReplaceTrigger`. The provided source code also includes source code for an OAA Facilitator Agent.

The source code appendix is offered solely as a means of further helping practitioners to construct a preferred embodiment of the invention. By no means is the source code intended to limit the scope of the present invention.

ILLUSTRATIVE APPLICATIONS

To further illustrate the technology of the preferred embodiment, we will next present and discuss two sample applications of the present inventions.

Unified Messaging

A further preferred embodiment of present invention incorporates a Unified Messaging application extending the Automated Office application presented previously herein with an emphasis on ubiquitous access and dynamic pre-

24

sensation of the information and services supported by the agent community. The agents used in this application are depicted in FIG. 12.

A hypothetical example of realistic dialog using a preferred embodiment of the present invention can provide insight into how systems may preferably be built using the present invention. In this scenario, the user, with only a telephone as an interface, is planning a trip to Boston where he will soon give a presentation. Capitalized sentences are phrases spoken by the user into the telephone and processed by a phone agent 452.

Responses, unless otherwise indicated, are spoken by the system using text-to-speech generation agent 454.

1.1 Welcome to SRI International. Please enter your user ID and password.

<User enters touchtone ID and password>

Good to see you again Adam Cheyer. I am listening to you.

Every user interface agent 408, including the telephone agent 452, should know the identity of its user. This information is used in resolving anaphoric references such as "me" and "I", and allows multiple user interfaces operated by the same user to work together.

1.2 WHAT IS TODAY'S SCHEDULE?

Here is today's schedule for Adam Cheyer:

At 10 am for 1 hour, meeting with Dave.

At 3 ; pm for 1 hour, presentation about software agents.

End of Schedule.

If the user is operating both a graphical user interface and a telephone, as described in conjunction with the Automated Office application, the result of this spoken request is to display a calendar window containing the current schedule. In this case, with no graphical display available, the GEN_NL agent 1202 is tasked to produce a spoken response that can be played over the phone. GEN_NL shares the same dynamic vocabulary and phrasal rules as the natural language parser DCG_NL 426, and contains strategies for producing responses to queries using either simple or list-based multimedia utterances.

1.3 FIND FRIDAY'S WEATHER IN BOSTON.

The weather in Boston for Friday is as follows:

Sunny in the morning. Partly cloudy in the afternoon with a 20 percent chance of thunderstorms late. Highs in the mid 70s.

In addition to data accessible from legacy applications, content may be retrieved by web-reading agents which provide wrappers around useful websites.

1.4 FIND ALL NEW MAIL MESSAGES.

There are 2 messages available.

Message 1, from Mark Tierny, entitled "OAA meeting."

1.5 NEXT MESSAGE

Message 2, from Jennifer schweffler, entitled "Presentation Summary."

1.6 PLAY IT.

This message is a multipart MIME-encoded message. There are two parts.

Part 1. (Voicemail message, not text-to-speech):

Thanks for taking part as a speaker in our conference.

The schedule will be posed soon on our homepage.

1.7 NEXT PART

Part 2. (read using text-to-speech).

Thanks for taking part as a speaker in our conference.

The schedule will be posted soon on our homepage.

US 7,069,560 B1

25

1.8 PRINT MESSAGE

Command executed.

Mail messages are no longer just simple text documents, but often consist of multiple subparts containing audio files, pictures, webpages, attachments and so forth. When a user asks to play a complex email message over the telephone, many different agents may be implicated in the translation process, which would be quite different given the request "print it." The challenge is to develop a system which will enable agents to cooperate in an extensible, flexible manner that alleviates explicit coding of agent interactions for every possible input/output combination.

In a preferred embodiment of the present invention, each agent concentrates only on what it can do and on what it knows, and leaves other work to be delegated to the agent community. For instance, a printer agent 1204, defining the solvable print(Object,Parameters), can be defined by the following pseudo-code, which basically says, "If someone can get me a document, in either POSTSCRIPT or text form, I can print it."

```
print(Object, Parameters) {
  'If Object is reference to "it", find an appropriate
  document
  if (Object="ref(it)")
    oaa_Solve(resolve_reference(the, document, Params,
  Object,[]));
  'Given a reference to some document, ask for the
  document in POSTSCRIPT
  if(Object="id(Pointer)")
    oaa_Solve(resolve_id_as(id(Pointer), postscript, [],
  Object,[]));
  'If Object is of type text or POSTSCRIPT, we can print it.
  if((Object is of type Text) or (Object is of type Postscript))
    do_print(Object);
}
```

In the above example, since an email message is the salient document, the mail agent 442 will receive a request to produce the message as POSTSCRIPT. Whereas the mail agent 442 may know how to save a text message as POSTSCRIPT, it will not know what to do with a webpage or voicemail message. For these parts of the message, it will simply send oaa_Solve requests to see if another agent knows how to accomplish the task.

Until now, the user has been using only a telephone as user interface. Now, he moves to his desktop, starts a web browser 436, and accesses the URL referenced by the mail message.

1.9 RECORD MESSAGE

Recording voice message. Start speaking now.

1.10 THIS IS THE UPDATED WEB PAGE CONTAINING THE PRESENTATION SCHEDULE

Message one recorded.

1.11 IF THIS WEB PAGE CHANGES, GET IT TO ME WITH NOTE ONE.

Trigger added as requested.

In this example, a local agent 436 which interfaces with the web browser can return the current page as a solution to the request "oaa_Solve(resolve_reference(this, web_page, [],Ref,[]))", sent by the NL agent 426. A trigger is installed on a web agent 436 to monitor changes to the page, and when the page is updated, the notify agent 446 can find the user and transmit the webpage and voicemail message using the most appropriate media transfer mechanism.

This example based on the Unified Messaging application is intended to show how concepts in accordance with the

26

present invention can be used to produce a simple yet extensible solution to a multi-agent problem that would be difficult to implement using a more rigid framework. The application supports adaptable presentation for queries across dynamically changing, complex information; shared context and reference resolution among applications; and flexible translation of multimedia data. In the next section, we will present an application which highlights the use of parallel competition and cooperation among agents during multi-modal fusion.

Multimodal Map

A further preferred embodiment of present invention incorporates the Multimodal Map application. This application demonstrates natural ways of communicating with a community of agents, providing an interactive interface on which the user may draw, write or speak. In a travel-planning domain illustrated by FIG. 13, available information includes hotel, restaurant, and tourist-site data retrieved by distributed software agents from commercial Internet sites. Some preferred types of user interactions and multimodal issues handled by the application are illustrated by a brief scenario featuring working examples taken from the current system.

Sara is planning a business trip to San Francisco, but would like to schedule some activities for the weekend while she is there. She turns on her laptop PC, executes a map application, and selects San Francisco.

2.1 [Speaking] Where is downtown?

Map scrolls to appropriate area.

2.2 [Speaking and drawing region] Show me all hotels near here.

Icons representing hotels appear.

2.3 [Writes on a hotel] Info?

A textual description (price, attributes, etc.) appears.

2.4 [Speaking] I only want hotels with a pool.
some hotels disappear.

2.5 [Draws a crossout on a hotel that is too close to a highway]

Hotel disappears

2.6 [Speaking and circling] Show me a photo of his hotel.
Photo appears.2.7 [Points to another hotel]
Photo appears.2.8 [Speaking] Price of the other hotel?
Price appears for previous hotel.2.9 [Speaking and drawing an arrow] Scroll down.
Display adjusted.2.10 [Speaking and drawing an arrow toward a hotel]
What is the distance from this hotel to Fisherman's Wharf?
Distance displayed.2.11 [Pointing to another place and speaking] And the distance to here?
Distance displayed.

Sara decides she could use some human advice. She picks up the phone, calls Bob, her travel agent, and writes Start collaboration to synchronize his display with hers. At this point, both are presented with identical maps, and the input and actions of one will be remotely seen by the other.

US 7,069,560 B1

27

3.1 [Sara speaks and circles two hotels]

Bob, I'm trying to choose between these two hotels. Any opinions?

3.2 [Bob draws an arrow, speaks, and points]

Well, this area is really nice to visit. You can walk there from this hotel.

Map scrolls to indicated area. Hotel selected.

3.3 [Sara speaks] Do you think I should visit Alcatraz?

3.4 [Bob speaks] Map, show video of Alcatraz.

Video appears.

3.5 [Bob speaks] Yes, Alcatraz is a lot of fun.

A further preferred embodiment of present invention generates the most appropriate interpretation for the incoming streams of multi-modal input. Besides providing a user interface to a dynamic set of distributed agents, the application is preferably built using an agent framework. The present invention also contemplates aiding the coordinate competition and cooperation among information sources, which in turn works in parallel to resolve the ambiguities arising at every level of the interpretation process: low-level processing of the data stream, anaphora resolution, cross-modality influences and addressee.

Low-level processing of the data stream: Pen input may be preferably interpreted as a gesture (e.g., 2.5: cross-out) by one algorithm, or as handwriting by a separate recognition process (e.g., 2.3: "info?"). Multiple hypotheses may preferably be returned by a modality recognition component.

Anaphora resolution: When resolving anaphoric references, separate information sources may contribute to resolving the reference: context by object type, deictic, visual context, database queries, discourse analysis. An example of information provided through context by object type is found in interpreting an utterance such as "show photo of the hotel", where the natural language component can return a list of the last hotels talked about. Deictic information in combination with a spoken utterance like "show photo of this hotel" may preferably include pointing, circling, or arrow gestures which might indicate the desired object (e.g., 2.7). Deictic references may preferably occur before, during, or after an accompanying verbal command. Information provided in a visual context, given for the request "display photo of the hotel" may preferably include the user interface agent might determine that only one hotel is currently visible on the map, and therefore this might be the desired reference object. Database queries preferably involving information from a database agent combined with results from other resolution strategies. Examples are "show me a photo of the hotel in Menlo Park" and 2.2. Discourse analysis preferably provides a source of information for phrases such as "No, the other one" (or 2.8).

The above list of preferred anaphora resolution mechanisms is not exhaustive. Examples of other preferred resolution methods include but are not limited to spatial reasoning ("the hotel between Fisherman's Wharf and Lombard Street") and user preferences ("near my favorite restaurant").

Cross-modality influences: When multiple modalities are used together, one modality may preferably reinforce or remove or diminish ambiguity from the interpretation of another. For instance, the interpretation of an arrow gesture may vary when accompanied by different verbal commands (e.g., "scroll left" vs. "show info about this hotel"). In the latter example, the system must take into account how accurately and unambiguously an arrow selects a single hotel.

28

Addressee: With the addition of collaboration technology, humans and automated agents all share the same workspace. A pen doodle or a spoken utterance may be meant for either another human, the system (3.1), or both (3.2).

The implementation of the Multimodal Map application illustrates and exploits several preferred features of the present invention: reference resolution and task delegation by parallel parameters of oaa_Solve, basic multi-user collaboration handled through built-in data management services, additional functionality readily achieved by adding new agents to the community, domain-specific code cleanly separated from other agents.

A further preferred embodiment of present invention provides reference resolution and task delegation handled in a distributed fashion by the parallel parameters of oaa_Solve, with meta-agents encoding rules to help the facilitator make context- or user-specific decisions about priorities among knowledge sources.

A further preferred embodiment of present invention provides basic multi-user collaboration handled through at least one built-in data management service. The map user interface preferably publishes data solvable for elements such as icons, screen position, and viewers, and preferably defines these elements to have the attribute "shareable". For every update to this public data, the changes are preferably automatically replicated to all members of the collaborative session, with associated callbacks producing the visible effect of the data change (e.g., adding or removing an icon).

Functionality for recording and playback of a session is preferably implemented by adding agents as members of the collaborative community. These agents either record the data changes to disk, or read a log file and replicate the changes in the shared environment.

The domain-specific code for interpreting travel planning dialog is preferably separated from the speech, natural language, pen recognition, database and map user interface agents. These components were preferably reused without modification to add multimodal map capabilities to other applications for activities such as crisis management, multi-robot control, and the MVEWS tools for the video analyst.

Improved Scalability and Fault Tolerance

Implementations of a preferred embodiment of present invention which rely upon simple, single facilitator architectures may face certain limitations with respect to scalability, because the single facilitator may become a communications bottleneck and may also represent a single, critical point for system failure.

Multiple facilitator systems as disclosed in the preferred embodiments to this point can be used to construct peer-to-peer agent networks as illustrated in FIG. 14. While such embodiments are scalable, they do possess the potential for communication bottlenecks as discussed in the previous paragraph and they further possess the potential for reliability problems as central, critical points of vulnerability to systems failure.

A further embodiment of present invention supports a facilitator implemented as an agent like any other, whereby multiple facilitator network topologies can be readily constructed. One example configuration (but not the only possibility) is a hierarchical topology as depicted in FIG. 15, where a top level Facilitator manages collections of both client agents 1508 and other Facilitators, 1504 and 1506. Facilitator agents could be installed for individual users, for a group of users, or as appropriate for the task.

Note further, that network work topologies of facilitators can be seen as graphs where each node corresponds to an

US 7,069,560 B1

29

instance of a facilitator and each edge connecting two or more nodes corresponds to a transmission path across one or more physical transport mechanisms. Some nodes may represent facilitators and some nodes may represent clients. Each node can be further annotated with attributes corresponding to include triggers, data, capabilities but not limited to these attributes.

A further embodiment of present invention provides enhanced scalability and robustness by separating the planning and execution components of the facilitator. In contrast with the centralized facilitation schemes described above, the facilitator system 1600 of FIG. 16 separates the registry/planning component from the execution component. As a result, no single facilitator agent must carry all communications nor does the failure of a single facilitator agent shut down the entire system.

Turning directly to FIG. 16, the facilitator system 1600 includes a registry/planner 1602 and a plurality of client agents 1612-1616. The registry/planner 1604 is typically replicated in one or more locations accessible by the client agents. Thus if the registry/planner 1604 becomes unavailable, the client agents can access the replicated registry/planner(s).

This system operates, for example, as follows. An agent transmits a goal 1610 to the registry planner 1602. The registry/planner 1604 translates the goal into an unambiguous execution plan detailing how to accomplish any sub-goals developed from the compound goal, as well as specifying the agents selected for performing the sub-goals. This execution plan is provided to the requesting agent which in turn initiates peer-to-peer interactions 1618 in order to implement the detailed execution plan, routing and combining information as specified within the execution plan. Communication is distributed thus decreasing sensitivity of the system to bandwidth limitations of a single facilitator agent. Execution state is likewise distributed thus enabling system operation even when a facilitator agent fails.

Further embodiments of present invention incorporate into the facilitator functionality such as load-balancing, resource management, and dynamic configuration of agent locations and numbers, using (for example) any of the topologies discussed. Other embodiments incorporate into a facilitator the ability to aid agents in establishing peer-to-peer communications. That is, for tasks requiring a sequence of exchanges between two agents, the facilitator assist the agents in finding one another and establishing communication, stepping out of the way while the agents communicate peer-to-peer over a direct, perhaps dedicated channel.

Further preferred embodiments of the present invention incorporate mechanisms for basic transaction management, such as periodically saving the state of agents (both facilitator and client) and rolling back to the latest saved state in the event of the failure of an agent.

What is claimed is:

1. A software-based, flexible computer architecture for communication and cooperation among distributed electronic agents, the architecture contemplating a distributed computing system comprising:

- a plurality of service-providing electronic agents;
- a distributed facilitator agent functionally distributed across at least two computer processes, the facilitator agent capable of bi-directional communications with the plurality of service-providing electronic agents, the facilitator agent including:

30

an agent registry that declares capabilities for each of the plurality of service-providing electronic agents currently active within the distributed computing environment; and

a facilitating engine operable to interpret a service request as a base goal, the facilitating engine further operable for generating a goal satisfaction plan associated with the base goal, wherein the goal satisfaction plan involves:

using reasoning to determine sub-goal requests based on non-syntactic decomposition of the base goal and using said reasoning to co-ordinate and schedule efforts by the service-providing electronic agents for fulfilling the sub-goal requests in a cooperative completion of the base goal; and

wherein the plurality of service-providing electronic agents and the distributed facilitator agent communicate using an interagent Communication Language (ICL), wherein the ICL includes:

a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events.

2. A software-based, flexible computer architecture as recited in claim 1 wherein the distributed facilitator agent includes a plurality of single process facilitator agents each executing within a separate computer process, each of the single process facilitator agents being bi-directionally coupled with at least one other single process facilitator agent.

3. A software-based, flexible computer architecture as recited in claim 2 wherein each single process facilitator agent has any necessary facilitating functionality, a specific single process facilitator including:

a specific agent registry that declares capabilities for each of the plurality of service-providing electronic agents currently active within the process wherein the specific single process is executing, the specific agent registry further declaring capabilities made available to the specific single process facilitator agent through the at least one other single process facilitator agent bi-directionally coupled with the specific single process facilitator agent; and

a specific facilitating engine operable to interpret a service request as a base goal, the specific facilitating engine further operable to determine sub goals required to complete the base goal; the specific facilitating engine further operable to select service providing agents best capable of completing the sub goal and assigning the sub goals thereto.

4. A software-based, flexible computer architecture as recited in claim 3 wherein at least two of the plurality of single process facilitator agents reside upon separate computer systems.

5. A computer architecture as recited in claim 1, wherein the ICL enables agents to perform queries of other agents, exchange information with other agents, and set triggers within other agents, the ICL further defined by an ICL syntax supporting compound goal expressions such that goals within a single request provided according to the ICL syntax may be coupled by a conjunctive operator, a disjunctive operator, a conditional execution operator, and a parallel disjunctive operator that indicates that disjunctive goals are to be performed by different agents.

6. A computer architecture as recited in claim 5, wherein the ICL is computer platform independent.

US 7,069,560 B1

31

7. A computer architecture as recited in claim 6 wherein the ICL is independent of computer programming languages in which the plurality of agents are programmed.

8. A computer architecture as recited in claim 7 wherein the ICL syntax supports explicit task completion constraints within goal expressions.

9. A computer architecture as recited in claim 8 wherein possible types of task completion constraints include use of specific agent constraints and response time constraints.

10. A computer architecture as recited in claim 8 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

11. A computer architecture as recited in claim 5 wherein the ICL syntax supports explicit task completion advisory suggestions within goal expressions.

12. A computer architecture as recited in claim 5 wherein each autonomous service-providing electronic agent defines and publishes a set of capability declarations or solvables, expressed in ICL, that describes services provided by such electronic agent.

13. A computer architecture as recited in claim 12 wherein an electronic agent's solvables define an interface for the electronic agent.

14. A computer architecture as recited in claim 1 wherein the distributed facilitator agent is formed in a hierarchical topology.

15. A computer architecture as recited in claim 14 wherein the hierarchical topology includes a top level facilitator agent and at least one other facilitator agent registered within the top level facilitator agent, the top level facilitator agent operable to directly manage those service-providing agents registered within the top level facilitator agent and indirectly manage those service-providing agents registered within the at least one other agent registered with the top level facilitator agent.

16. A computer architecture as recited in claim 15 wherein the top level facilitator agent and the at least one other facilitator agent are executing on different computer systems.

17. A computer architecture as recited in claim 15 wherein the at least one other facilitator agent is installed for a specific computer user.

18. A computer architecture as recited in claim 15 wherein the at least one other facilitator agent is installed for a specific group of users.

19. A computer architecture as recited in claim 15 wherein the at least one other facilitator agent is installed for a specific computer application.

20. A computer architecture as recited in claim 1 wherein the distributed facilitator agent includes a planning component executing within a first computer process and an execution component executing within a second computer process.

21. A computer architecture as recited in claim 20 wherein the planning component is one of a plurality of synchronized planning components each executing with separate computer processes, whereby the computer architecture provides a more robust operating environment due to redundancy of the planning component functionality of the distributed facilitator agent.

22. A software-based flexible computer architecture for communication and cooperation among distributed electronic agents, the architecture contemplating a distributed computing system comprising:

- a plurality of service providing electronic agents;
- at least one facilitator agent capable of receiving a service requests in the form of a base goal from a service-

32

requesting agent in an interagent communication language (ICL) and capable of determining sub goals necessary to accomplish the base goal, the facilitator agent operable to allocate each sub-goal to at least one service-providing agent capable of accomplishing the sub-goal as determined by the registry, the facilitator agent being distinct from service-providing agents, wherein the ICL includes a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events; and

at least one service-requesting agent capable of making a request directly to a service-providing agent as a peer to peer communication for accomplishment of at least one of the sub-goals.

23. A software based, flexible computer system as recited in claim 22 wherein the peer to peer communication is in a language other than an interagent communication language.

24. A software based, flexible computer system as recited in claim 22 wherein the peer to peer communication is bi-directional.

25. A software based, flexible computer system as recited in claim 22 wherein the agent operable to make said peer to peer service request is said facilitator agent.

26. A distributed facilitator agent functionally distributed across at least two computer processes, the distributed facilitator agent arranged to coordinate cooperative task completion within a distributed computing environment having a plurality of autonomous service-providing electronic agents, the distributed facilitator agent comprising:

an agent registry that declares capabilities of service-providing electronic agents currently active within the distributed computing environment; and

a facilitating engine operable to parse a service request in order to interpret a compound goal set forth therein, the service request formed according to an Interagent Communication Language (ICL), the ICL including a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events, the facilitating engine further operable to generate a goal satisfaction plan associated with the compound goal, wherein the goal satisfaction plan involves:

using reasoning to determine sub-goal requests based on non-syntactic decomposition of the base goal and using said reasoning to co-ordinate and schedule efforts by the service-providing electronic agents for fulfilling the sub-goal requests in a cooperative completion of the base goal.

27. A facilitator agent as recited in claim 26, wherein the facilitating engine is capable of modifying the goal satisfaction plan during execution, the modifying initiated by events such as new agent declarations within the agent registry, decisions made by remote agents, and information provided to the facilitating engine by remote agents.

28. A facilitator agent as recited in claim 26 wherein the agent registry includes a symbolic name, a unique address, data declarations, trigger declarations, task declarations, and process characteristics for each active agent.

29. A facilitator agent as recited in claim 26 wherein the facilitating engine is operable to install a trigger mechanism requesting that a certain action be taken when a certain set of conditions are met.

30. A facilitator agent as recited in claim 29 wherein the trigger mechanism is a communication trigger that monitors

US 7,069,560 B1

33

communication events and performs the certain action when a certain communication event occurs.

31. A facilitator agent as recited in claim 29 wherein the trigger mechanism is a data trigger that monitors a state of a data repository and performs the certain action when a certain data state is obtained.

32. A facilitator agent as recited in claim 31 wherein the data repository is local to the facilitator agent.

33. A facilitator agent as recited in claim 31 wherein the data repository is remote from the facilitator agent.

34. A facilitator agent as recited in claim 29 wherein the trigger mechanism is a task trigger having a set of conditions.

35. A facilitator agent as recited in claim 26, the facilitator agent further including a global database accessible to at least one of the service-providing electronic agents.

36. A facilitator agent as recited in claim 26 wherein the distributed facilitator agent includes a plurality of single process facilitator agents each executing within a separate computer process, each of the single process facilitator agents being bi-directionally coupled with at least one other single process facilitator agent.

37. A facilitator agent as recited in claim 36 wherein each single process facilitator agent has any necessary facilitating functionality, a specific single process facilitator including:

a specific agent registry that declares capabilities for each of the plurality of service-providing electronic agents currently active within the process wherein the specific single process is executing, the specific agent registry further declaring capabilities made available to the specific single process facilitator agent through the at least one other single process facilitator agent bi-directionally coupled with the specific single process facilitator agent; and

a specific facilitating engine operable to parse a service request in order to interpret an arbitrarily complex goal set forth therein, the specific facilitating engine further operable to construct a goal satisfaction plan using reasoning to determine sub-goal requests based on non-syntactic decomposition of the base goal and using said reasoning to co-ordinate and schedule efforts by the service-providing electronic agents for fulfilling the sub-goal requests in a cooperative completion of the base goal.

38. A facilitator agent as recited in claim 37 wherein at least two of the plurality of single process facilitator agents reside upon separate computer systems.

39. A facilitator agent as recited in claim 38 wherein the distributed facilitator agent is formed in a hierarchical topology.

40. A facilitator agent as recited in claim 39 wherein the hierarchical topology includes a top level facilitator agent and at least one other facilitator agent registered within the top level facilitator agent, the top level facilitator agent operable to directly manage those service-providing agents registered within the top level facilitator agent and indirectly manage those service-providing agents registered within the at least one other agent registered with the top level facilitator agent.

41. A facilitator agent as recited in claim 40 wherein the top level facilitator agent and the at least one other facilitator agent are executing on different computer systems.

42. A facilitator agent as recited in claim 40 wherein the at least one other facilitator agent is installed for a specific computer user.

34

43. A facilitator agent as recited in claim 40 wherein the at least one other facilitator agent is installed for a specific group of users.

44. A facilitator agent as recited in claim 40 wherein the at least one other facilitator agent is installed for a specific computer application.

45. A facilitator agent as recited in claim 26 wherein the distributed facilitator agent includes a planning component executing within a first computer process and an execution component executing within a second computer process.

46. A facilitator agent as recited in claim 45 wherein the planning component is one of a plurality of synchronized planning components each executing with separate computer processes, whereby the computer architecture provides a more robust operating environment due to redundancy of the planning component functionality of the distributed facilitator agent.

47. A facilitator agent for coordinating cooperative task completion within a distributed computing environment having a plurality of autonomous service-providing electronic agents, the facilitator agent comprising:

a registry of capabilities of the service-providing electronic agents; and

a facilitating engine operable to determine a set of sub goals necessary to accomplish a base goal, and then allocate such sub-goals to those agents capable of accomplishing the sub-goals as determined by the registry, said facilitating agent further capable of initiating a direct peer to peer communication between a service-requesting agent and a service-providing agent of at least one sub-goal, and said facilitating agent being distinct from service-providing agents; and

wherein the service-providing agent and the service-requesting agent communicate using an Interagent Communication Language (ICL), the ICL includes a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events.

48. A facilitator as set forth in claim 47 wherein the facilitator agent is functionally distributed across at least two electronic agents.

49. A facilitator as set forth in claim 48 wherein the peer to peer communication is between said distributed facilitator agents.

50. A computer-implemented method for providing cooperative task completion within a distributed computing environment, the distributed computing environment including a plurality of autonomous electronic agents, the distributed computing environment supporting an Interagent Communication Language, the computer-implemented method comprising the acts of:

providing a plurality of synchronized agent registries each declaring capabilities of service-providing electronic agents currently active within the distributed computing environment, the plurality of synchronized agent registries each resident within a separate computer process;

interpreting a service request in order to determine a base goal, the service request adhering to the Interagent Communication Language (ICL), the ICL including a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events, the act of interpreting including the sub-acts of:

US 7,069,560 B1

35

determining a goal satisfaction plan that is associated with the base goal;

wherein the goal satisfaction plan involves:

using reasoning to determine sub-goal requests based on non-syntactic decomposition of the base goal and using said reasoning to co-ordinate and schedule efforts by the service-providing electronic agents for fulfilling the sub-goal requests in a cooperative completion of the base goal; and implementing the base goal satisfaction plan.

51. A computer implemented process as recited in claim 50 wherein the step of interpreting a service request is controlled by a computer process separate from the computer processes wherein the plurality of synchronized agent registries reside.

52. A computer implemented process for providing coordinated task completion within a distributed computing environment, the distributed computing environment including a plurality of autonomous electronic agents, the computer implemented method comprising the steps of:

providing at least one agent registry including capabilities of service providing electronic agents;

interpreting a service request in the form of a base goal, the service request being in a interagent communication language (ICL), the ICL including a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the Parameter lists further refine the one or more events;

determining a plurality of sub goals necessary to accomplish the base goal;

selecting from said registry at least one service providing agent capable of completing said sub goals;

delegating at least one sub goal as a peer to peer service request directly from a service requesting agent to a service providing agent; and

delegating any remaining sub goals as service request in the interagent communication language to the selected agents capable of completing the remaining sub-goals.

36

53. A computer-implemented method for providing cooperative task completion within a distributed computing environment, the distributed computing environment including a plurality of autonomous electronic agents, the distributed computing environment supporting a dynamically expandable Interagent Communication Language ("ICL"), the computer implemented method comprising the acts of:

providing a plurality of synchronized agent registries each declaring a set of functional capabilities for one or more of the autonomous service-providing electronic agents, each of the synchronized agent registries being resident within a separate computer process;

receiving a service request adhering to the ICL, the ICL including a layer of conversational protocol defined by event types and parameter lists associated with one or more of the events, wherein the parameter lists further refine the one or more events;

determining a base goal based on the service request; determining a delegation plan wherein the delegation plan involves:

using reasoning to determine sub-goal requests based on non-syntactic decomposition of the base goal and using said reasoning to co-ordinate and schedule efforts by the autonomous service-providing electronic agents for fulfilling the sub-goal requests in a cooperative completion of the base goal; and implementing the delegation plan.

54. A computer implemented method as recited in claim 53 wherein the act of providing a plurality of agent registries further includes replicating at least a portion of one or more of the synchronized agent registries across a plurality of separate computer processes.

55. A computer implemented method as recited in claim 53 wherein the act of implementing the delegation plan is controlled by a computer process separate from the computer processes wherein the plurality of synchronized agent registries reside.

* * * * *

Exhibit 3

SEALED

**REDACTED IN
ITS ENTIRETY**

Exhibit 4

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

IPA TECHNOLOGIES INC.,

Plaintiff,

v.

AMAZON.COM, INC., and AMAZON
DIGITAL SERVICES, LLC,

Defendants.

Civil Action No. 1:16-cv-01266-RGA

DECLARATION OF DAVID L. MARTIN

I, David L. Martin, declare as follows:

1. I am over 18 years of age and competent to make this declaration. All statements set forth herein are true and correct and are based on my personal knowledge. If called and sworn as a witness, I could and would testify as set forth below. Furthermore, if necessary, I am willing to testify at trial to the matters set forth below.

2. I was employed by SRI International ("SRI") from March of 1994 to March 2010 as a computer scientist. My regular job duties included research, software development, project management, interfacing with SRI clients and collaborators on SRI-related software development, providing content to and updating portions of SRI's website (<http://www.ai.sri.com/>), authoring technical papers, and presenting various research and software development presentations at technology conferences. As a part of my regular duties, I was responsible for storing, maintaining and updating software-related files, including source code, on SRI's various servers.

3. In 1994, I started working on SRI's Open Agent Architecture project, also referred to as "OAA." My responsibilities on the OAA project included research and software development.

4. I, along with Adam J. Cheyer, played major roles in the creation and development

of OAA.

5. Mr. Cheyer and I are the sole named inventors on U.S. Patent No. 6,851,115, titled “Software-based Architecture for Communication and Cooperation Among Distributed Electronic Agents” (“the ’115 Patent”) and 7,069,560, titled “Highly Scalable Software-based Architecture for Communication and Cooperation Among Distributed Electronic Agents” (“the ’560 Patent”). Both of these patents relate to our work on OAA, specifically OAA 2.

6. SRI referred to the first generation of OAA as “OAA version 1” or “OAA 1” and the second generation of OAA as “OAA version 2” or “OAA 2.”

7. Both the OAA 1 and OAA 2 source code were frequently revised, which resulted in several source code versions within both OAA 1 and OAA 2.

8. OAA 1 source code was first publicly released in November 1998 and work on OAA 1 continued after that time.

9. Work on OAA 2 began around the middle of 1996.

10. OAA 1 had limited capabilities and, among other things, could not process “arbitrarily complex goals,” whereas OAA 2 could.

11. I, along with Adam Cheyer, developed, wrote, and implemented the OAA 2 software code that led to the inventions disclosed in the ’115 and ’560 patents.

12. While I was working on OAA research and software development, I saved, stored, updated, and maintained the OAA software code on SRI’s servers, including SRI’s “trestle3” server.

13. While I was working on OAA, I provided OAA content and updates to SRI’s website, <http://www.ai.sri.com/>.

14. To my knowledge, no OAA source code files were publicly displayed or linked on this website, or made available for download from this website, until after the respective OAA

versions were released to the public, if at all.

15. During my employment at SRI, I did not track the use of the OAA source code and text files. To my knowledge, no one at SRI did this.

16. During my employment at SRI, I did not keep records of which OAA source code and text files were used or implemented in an application for any particular demonstration, video presentation, or associated with any paper, article or presentation. To my knowledge, no one at SRI did this.

17. Using the “last modified” dates found in the source code file detail information was not a naming convention I used for the OAA source code and text files. To my knowledge, no one at SRI did this.

18. During my employment at SRI, I did not adhere to any particular naming conventions for the OAA source code and text files or their file directories. To my knowledge, no one at SRI did this.

19. During my employment at SRI leading up to the 1999 filing date of the '115 Patent, I took great care not to disclose OAA 2 source code files to the public. To my knowledge, no one at SRI disclosed OAA 2 source code files prior to that date.

20. I specifically recall that the compound.pl file, which was written for OAA 2 and provided additional functionality, was held back from the public and kept confidential until at least the 1999 filing date of the '115 Patent.

21. The compound.pl code was under development for a long period of time and, during that period, it was kept in a location separate from the main body of the OAA source code and separate from any OAA-based applications.

22. The source code files Amazon titled “1997 compound.pl” resided in a single location in the “trestle3” production made by SRI in this case.

23. The compound.pl file included in the source code files Amazon titled “1997 compound.pl” was not a completed version of this file.

24. The version of the compound.pl code included with the ’115 Patent application was not made public until sometime after January 5, 1999. No version of the source code file compound.pl was made public prior to this date.

25. I withheld the compound.pl code from the public and kept it confidential even after the ’115 Patent application was filed. To my knowledge, everyone at SRI did so.

26. For the compound.pl code, like other OAA code under development, there was no particular organizational scheme by which the location of this compound.pl code was selected.

27. I was provided access to the OAA source code SRI produced in this lawsuit on September 4, 2019. The compound.pl code that Amazon is relying on for its invalidity theory based on OAA only resides in a single location in the “trestle3” production (the agents/compound folder, which contains four copies of compound.pl), and there are no files in that location related to the construction of any application systems.

28. I examined the application systems residing in the “trestle3” source code production. The compound.pl code that Amazon is relying on for its invalidity theory based on OAA was not used in any of them.

29. I also reviewed the OAA-trestle3\ directory. My review established that at least two folders referenced in the README.TXT file in the OAA-trestle3\directory are not found in the OAA-trestle3\ directory.

30. As a matter of practice, the most up-to-date version of OAA was not always incorporated into demonstrations.

31. The concepts described in papers and presentations related to OAA had not always been implemented in code by the publication or presentation date.

32. I have reviewed the references Amazon is relying upon for their invalidity case based on OAA.

33. The source code Amazon titles “1994/1995 Automated Office” is based on OAA 1.

34. The source code Amazon titles “1996 Automated Office” is based on OAA 1.

35. The source code Amazon titles “1996 Multimodal Maps” is based on OAA 1.

36. The source code Amazon titles “1997 compound.pl” includes an incomplete version of the compound.pl file, which would eventually become part of OAA 2. It also includes a version of the agent.pl file, which had been modified as part of OAA 2 development, and which contains changes that were made specifically for OAA 2.

37. The article Amazon titles the “Martin paper” is based on OAA 2.

38. The article Amazon titles the “Multimodal Maps paper” is based on OAA 1.

39. The article Amazon titles the “MVEWS paper” is based on OAA 2.

40. The presentation slides Amazon titles the “PAAM 98 presentation” is based on OAA 2.

41. The article Amazon titles the “InfoWorld article” is based on OAA 1.

42. The video Amazon titles “CNET Demonstration of Automated Office” is based on OAA 1.

43. The video Amazon titles “SRI Demonstration of Automated Office” is based on OAA 1.

44. The video Amazon titles “Video Demonstration of OAA CARS” is based on OAA 1.

45. The article Amazon titles the “Julia paper” is likely based on OAA 1, though I cannot be certain based on the content of the article.

46. During any demonstrations of OAA that I presented or attended before January 5, 1998, I never relinquished control of OAA to the persons attending the demonstration—nor did any SRI presenter.

47. During any demonstrations of OAA that I presented or attended before January 5, 1998, I never left OAA in the possession of the persons attending the demonstration—nor did any SRI presenter.

48. During any demonstrations of OAA that I presented or attended before January 5, 1998, I never allowed the persons attending the demonstration to handle OAA or directly request that it perform any particular task—nor did any SRI presenter.

49. SRI never charged persons to attend demonstrations of OAA.

50. No products were sold, nor orders taken or solicited, during any demonstrations of OAA.

Under penalty of perjury, I declare that the foregoing is true and correct to the best of my knowledge, information, and belief. I further state that these statements are made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both under Section 1001 of Title 18 of the U.S. Code.

Executed on this 24 day of June 2021.

A handwritten signature in blue ink that reads "David L. Martin". The signature is written in a cursive style and is positioned above a horizontal line.

David. L. Martin

Exhibit 5

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

IPA TECHNOLOGIES INC.,

Plaintiff,

v.

AMAZON.COM, INC., and AMAZON
DIGITAL SERVICES, LLC,

Defendants.

Civil Action No. 1:16-cv-01266-RGA

DECLARATION OF C. RAYMOND PERRAULT

I, C. Raymond Perrault, declare as follows:

1. I am over 18 years of age and competent to make this declaration. My knowledge of the matters stated within this declaration is either personal or has been developed and confirmed through discussions with other SRI employees or SRI records. Further, I am authorized to testify on behalf of SRI to the matters set forth herein. If called and sworn as a witness, I could and would testify as set forth below. Furthermore, if necessary, I am willing to testify at trial to the matters set forth below.

2. I have been employed by SRI International (“SRI”) since 1993. My current title is Distinguished Computer Scientist, and my regular job duties include project management and business development.

3. The Open Agent Architecture (“OAA”) was a project created and developed at SRI.

4. OAA was a framework for integrating a community of heterogenous software agents in a distributed environment.

5. At least two versions of OAA were developed at SRI and eventually released to the public: OAA 1 and OAA 2.

6. Internal work on OAA 1 started around 1993 and SRI created a working version of the Automated Office application that year, built using OAA 1.

7. Work on OAA 1 and its applications continued and the technology evolved over the next few years.

8. OAA 1 was released to the public on November 20, 1998 and components of OAA 1 continued to be added through 1999. This is confirmed and evidenced by Exhibit 6 to this declaration, which accurately reflects the release date of OAA 1.

9. Internal work on OAA 2 started around the middle of 1996.

10. OAA 2 was a significant rewrite and cleanup of OAA 1, and it offered a number of significant improvements and new features.

11. Among other things, OAA 2 provided a simpler, more elegant programming model; more expressive capabilities declarations; new parameters, including advice parameters; better organization and support for triggers; support for instantiation of meta-agents; support for compound goals; better structured code; and object-based representations of ICL expressions.

12. OAA 2 was better documented and supported than OAA 1 and the capabilities and design of OAA 2 embodied significant changes and improvements over OAA 1.

13. OAA 2 was released to the public on November 15, 1999 and additional sub-versions of OAA 2 were released through at least 2007. This is confirmed and evidenced by Exhibit 2 to this declaration, which accurately reflects the release date of OAA 2.

14. No later than October 29, 1996, SRI created and maintained a website related to OAA: ai.sri.com/~oaa/. That website includes tabs called About OAA, News!, Applications, Publications, Documentation & Distribution, and Contact and Community.

15. The purpose of the OAA website was to provide general, high-level information about OAA.

16. On the SRI website, SRI instructed developers to work with OAA 2 unless they needed OAA 1 to support an existing system or reuse existing agents based on OAA 1. Evidence of this is seen in Exhibit 6 to this declaration.

17. The SRI website also implied that the only reason for a developer to consider using OAA 1 would be because it provides libraries for more programming languages. Evidence of this is seen in Exhibit 4 to this declaration.

18. SRI created several applications based on OAA prior to January 5, 1998, including Automated Office (which utilized agents to automate access to databases, calendars, and other work tools) and Multimodal Maps (which provided a map-style interface to allow users to retrieve information and issue commands to the distributed agents).

19. SRI no longer has in its possession, custody, or control any emails sent, received, or exchanged between the SRI employees working on the OAA project (or between SRI employees and third parties) in the 1990s.

20. SRI is the original assignee of patents related to OAA work, including U.S. Patent Nos. 6,851,115, entitled “Software-Based Architecture for Communication and Cooperation Among Distributed Electronic Agents” and 7,069,560, entitled “Highly Scalable Software-Based Architecture for Communication and Cooperation Among Distributed Electronic Agents” (collectively, “the Asserted Patents”).

21. The inventors listed on the Asserted Patents, Adam J. Cheyer and David L. Martin, were SRI employees at the time the original patent application was filed on January 5, 1999.

22. Mr. Cheyer and Mr. Martin were two of the key scientists involved in creating and developing OAA during their time at SRI.

23. On July 17, 2019, Amazon served SRI with a subpoena to produce certain documents and information. SRI served objections to the subpoena on July 23, 2019 and produced

documents, including source code and files related to OAA, on September 4, 2019 (“SRI OAA Source Code”).

24. The SRI OAA Source Code produced in this litigation was never maintained or stored according to any standardized or organizational method, nor did SRI track its usage.

25. SRI did not adhere to any particular naming conventions for the SRI OAA Source Code or associated text files and their file directories.

26. Much of the SRI OAA Source Code includes file information that indicates dates in the 1990s, but SRI cannot verify the accuracy of these dates.

27. SRI also cannot verify who had access to or may have modified the SRI OAA Source Code-related files or their file locations.

28. SRI is unable to determine which particular SRI OAA Source Code files were used in any particular demonstrations of OAA, if any.

29. SRI did not maintain any records showing whether any of the SRI OAA Source Code and associated text files were ever used or implemented in an application for any particular demonstration, video presentation, or associated with any paper, article or presentation.

30. SRI is unable to determine which, if any, of the SRI OAA Source Code and associated text files were ever used or implemented in an application for any particular demonstration, video presentation, or associated with any paper, article or presentation.

31. SRI did not maintain any records showing whether any of the SRI OAA Source Code and associated text files were ever disclosed to the public

32. Exhibit 1 to this declaration is a true and correct copy of the home page of the Open Agent Architecture website (ai.sri.com/~oaa/) as of June 21, 2021 and (a) the portions cited in IPA’s brief were created at or near the same time as the OAA project by an SRI employee or employees with knowledge; (b) was made as part of the regular practice of SRI’s business

activities; and (c) has been kept in SRI's ordinary course of business.

33. Exhibit 2 to this declaration is a true and correct copy of the News tab of the Open Agent Architecture website (ai.sri.com/~oaa/) as of June 21, 2021 and (a) the portions cited in IPA's brief were created at or near the same time as the OAA project by an SRI employee or employees with knowledge; (b) was made as part of the regular practice of SRI's business activities; and (c) has been kept in SRI's ordinary course of business.

34. Exhibit 3 to this declaration is a true and correct copy of the Applications tab of the Open Agent Architecture website (ai.sri.com/~oaa/) as of June 21, 2021 and (a) the portions cited in IPA's brief were created at or near the same time as the OAA project by an SRI employee or employees with knowledge; (b) was made as part of the regular practice of SRI's business activities; and (c) has been kept in SRI's ordinary course of business.

35. Exhibit 4 to this declaration is a true and correct copy of the Documentation and Distribution tab of the Open Agent Architecture website (ai.sri.com/~oaa/) as of June 21, 2021 and (a) the portions cited in IPA's brief were created at or near the same time as the OAA project by an SRI employee or employees with knowledge; (b) was made as part of the regular practice of SRI's business activities; and (c) has been kept in SRI's ordinary course of business.

36. Exhibit 5 to this declaration is a true and correct copy of the page entitled "What's New in OAA 2?", linked to the Open Agent Architecture website (ai.sri.com/~oaa/) as of June 21, 2021 and (a) the portions cited in IPA's brief were created at or near the same time as the OAA project by an SRI employee or employees with knowledge; (b) was made as part of the regular practice of SRI's business activities; and (c) has been kept in SRI's ordinary course of business.

37. Exhibit 6 to this declaration is a true and correct copy of the page entitled "The Open Agent Architecture 1.0 Distribution," linked to the Open Agent Architecture website (ai.sri.com/~oaa/distribution/distribv1/) as of June 21, 2021 and (a) the portions cited in IPA's brief

were created at or near the same time as the OAA project by an SRI employee or employees with knowledge; (b) was made as part of the regular practice of SRI's business activities; and (c) has been kept in SRI's ordinary course of business.

38. Exhibit 7 to this declaration is a true and correct copy of the OAA Contact and Community tab of the OAA Agent Architecture website (ai.sri.com/~oaa/) as of June 21, 2021 and (a) the portions cited in IPA's brief were created at or near the same time as the OAA project by an SRI employee or employees with knowledge; (b) was made as part of the regular practice of SRI's business activities; and (c) has been kept in SRI's ordinary course of business.

Under penalty of perjury, I declare that the foregoing is true and correct to the best of my knowledge, information, and belief. I further state that these statements are made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both under Section 1001 of Title 18 of the U.S. Code.

Executed on this 22nd day of June 2021.



C. Raymond Perrault

C. Raymond Perrault

Exhibit 6

SEALED

**REDACTED IN
ITS ENTIRETY**

Exhibit 7

SEALED

**REDACTED IN
ITS ENTIRETY**

Exhibit 8

SEALED

**REDACTED IN
ITS ENTIRETY**

Exhibit 9

SEALED

**REDACTED IN
ITS ENTIRETY**

Exhibit 10

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

IPA TECHNOLOGIES, INC.,

Plaintiff,

v.

AMAZON.COM, INC., and
AMAZON DIGITAL SERVICES LLC,

Defendants.

The Honorable Richard G. Andrews

Civil Action No. 1:16-CV-01266-RGA

DECLARATION OF LUC JULIA

I, Luc Julia, hereby declare as follows:

1. I have personal knowledge of the facts in this declaration.
2. I currently work for Samsung Electronics as the CTO & Senior Vice President of Innovation. I was deposed in the above captioned case on December 1, 2020 related to work that I did at SRI International. During my deposition I testified about a CARS implementation of Open Agent Architecture (OAA) in 1996 that was funded in part by BMW. A true and correct copy of the 1996 CARS demonstration I discussed during my deposition is available at the link <http://www.adam.cheyer.com/videos/CARS-TravelMATE.ram> and is attached to this declaration as Exhibit A. During my deposition I also discussed a BBC documentary filmed and aired in 1997 that also covered the work I did on the CARS implementation of OAA.
3. During my deposition I referred to my website that lists 38 patents which name me as an inventor. A true and correct copy of the “patents and publications” page of my website

that identifies the patents on which I am named as an inventor is available at the link <http://lucjulia.com/patents.html> and is attached to this declaration as Exhibit B.

I declare under penalty of perjury under the laws of the United States that the foregoing is true and correct.

Executed on this 3rd day of December 2020 in Palo Alto, California

A handwritten signature in black ink, appearing to read 'Luc JULIA', with a stylized, overlapping loop.

Luc JULIA

Exhibit

A

Slip Sheet

Exhibit A

CARS-TravelMATE.ram

AMZ_IPA_00256440

Exhibit B

12/1/2020

Luc JULIA - Patents & Publications



Dr. Luc JULIA
The Future of Technology. Now.



Home

Links

Patents & Publications**Books (3)**

2020 - L'Intelligence Artificielle n'existe pas, J'ai Lu. (Paperback)



2020 - There is no such thing as Artificial Intelligence, First Editions.



2019 - L'Intelligence Artificielle n'existe pas, First Editions.

**Opinion Pieces in Newspapers or Magazines (11)****2020**11/12 - La maison connectée, auberge espagnole de la tech mondiale - [Challenges](#)10/15 - Côté IA, Elon Musk a moins de succès qu'avec sa Tesla - [Challenges](#)09/17 - TikTok et Toc Toc sont dans un bateau... - [Challenges](#)07/02 - La Silicon Valley sort grandie de la crise sanitaire - [Challenges](#)06/04 - La planète avance en ordre dispersé sur la traçabilité - [Challenges](#)

12/1/2020

Luc JULIA - Patents & Publications

05/07 - L'horripilant casse-tête des machines parlantes - [Challenges](#)
 04/16 - L'IA post COVID-19 - [Challenges](#)
 04/09 - Sur les réseaux sociaux aussi, gardez vos distances ! [Challenges](#)
 03/12 - L'industrie de l'e-sport draine beaucoup d'argent... et d'arnaques - [Challenges](#)
 02/13 - Prêts à chanter des Quantiques ? - [Challenges](#)
 01/16 - Ayez confiance ! - [Challenges](#)

Granted Patents (38)

2019

[10,360,202 - Agnostic data broker](#)
[10,334,024 - Electronic communication device](#)
[10,250,590 - Multi-factor device registration for establishing secure communication](#)

2018

[10,075,555 - System and method for delivering content to users on a network](#)

2017

[9,786,199 - System and method for assisting language learning](#)

2016

[9,374,805 - System and method for combining memory resources for use on a personal network](#)
[9,282,146 - System and method for delivering content to users on a network](#)
[9,247,110 - System and method for remotely monitoring and/or viewing images from a camera or video device](#)

2015

[9,166,879 - System and method for enabling the establishment and use of a personal network](#)
[9,077,766 - System and method for combining memory resources for use on a personal network](#)
[8,973,072 - System and method for programmatic link generation with media delivery](#)

2014

[8,880,677 - System and method for delivering content to users on a network](#)
[8,819,140 - System and method for enabling the establishment and use of a personal network](#)
[8,787,164 - Media delivery system and method for transporting media to desired target devices](#)
[8,755,567 - System and method for remotely monitoring and/or viewing images from a camera or video device](#)
[8,751,634 - System and method for delivering content to users on a network](#)
[8,738,730 - System and method for remotely controlling network resources](#)
[8,738,693 - System and method for managing distribution of media files](#)
[8,732,337 - System and method for delivering content to users on a network](#)

2013

[8,363,898 - System and method for remotely monitoring and/or viewing images from a camera or video device](#)

2012

[8,326,879 - System and method for enabling search and retrieval operations to be performed for data items and records using data obtained from associated voice files](#)
[8,281,037 - System and method for delivering content to users on a network](#)
[8,272,874 - System and method for assisting language learning](#)
[8,221,126 - System and method for performing programmatic language learning tests and evaluations](#)

[8,195,765 - System and method for remotely controlling network resources](#)

[8,195,744 - File sharing system for use with a network](#)

2011

12/1/2020

Luc JULIA - Patents & Publications

[8,068,642 - System and method for remotely monitoring and/or viewing images from a camera or video device](#)

[8,033,831 - System and method for programmatically evaluating and aiding a person learning a new language](#)

[7,937,484 - System and method for remotely controlling network resources](#)

2009

[7,627,138 - System and method for remotely monitoring and/or viewing images from a camera or video device](#)

[7,574,453 - System and method for enabling search and retrieval operations to be performed for data items and records using data obtained from associated voice files](#)

2007

[7,289,960 - System and method for speech activated internet browsing using open vocabulary enhancement](#)

[7,222,073 - System and method for speech activated navigation](#)

2006

[7,036,128 - Using a community of distributed electronic agents to support a highly mobile, ambient computing environment](#)

2004

[6,757,718 - Mobile navigation of network-based electronic information using spoken input](#)

[6,742,021 - Navigating network-based electronic information using spoken input with multimodal error feedback](#)

2003

[6,523,061 - System, method, and article of manufacture for agent-based navigation in a speech-based data navigation system](#)

[6,513,063 - Accessing network-based electronic information through scripted online interfaces using spoken input](#)

Chapters in Books (11)

2020 - La révolution des assistants vocaux, Oxana Gouliaéva, Eric Dosquet, Yvon Moysan, Dunod, Préface.



2019 - Data Éthique / IA Éthique : les deux visages d'un futur responsable, Business & Decision, P. 17.



2019 - The Hundred-Page Machine Learning Book 한국어로 (Korean version), Andriy Burkov, Foreword.

12/1/2020

Luc JULIA - Patents & Publications



2019 - The Hundred-Page Machine Learning Book en français (French version), Andriy Burkov, Avant-propos.



2019 - Guide pratique de l'Intelligence artificielle dans l'entreprise, Stéphane Roder, Editions Eyrolles, Préface.



2018 - L'intelligence artificielle expliquée à mon boss, Pierre Blanc, Editions Kawa, p. 264.



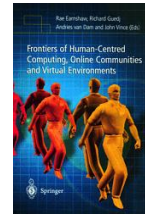
2001 - Cooperative Multimodal Communication - #2155, Bunt and Beun (Eds.), Springer, pp. 234-242.



2001 - Frontiers of Human-Centred Computing, Online Communities and Virtual Environments, Earnshaw/Guedj/Van Dam/Vince (Eds.), Springer, pp 190-197.

12/1/2020

Luc JULIA - Patents & Publications



1999 - Gesture-Based Communication in Human-Computer Interaction -#1739, Braffort/Gherbi/Gibet/Richardson/Teil (Eds.), Springer.




1998 - Collective Robotics - #1456, Drogoul/Tambe/Fukuda (Eds.), Springer, pp. 38-47.




1998 - Multimodal Human-Computer Communication - #1374, Bunt/Beun/Borghuis (Eds.), Springer, pp. 111-121.




Journals Articles (7)

ELSNNews - Summer 2001 - Number 10.2 - pp. 3-4. *ReadIt! Applies a Revolutionary Approach to Reading Education* 


Etats-Unis Microelectronique - January 2001 - Number 23 - pp. 19-20. *Les Reconnaissances de la Parole, de l'Ecriture et de Tous les Autres Signaux que le Genre Humain peut Produire pour Communiquer avec ses Semblables* 

Communications of the ACM - February 2000 - Volume 43 - Number 2 - pp. 57-63. *Maestro: Conductor of Multimedia Analysis Technologies* 

Virtual Reality - Winter 1999 - Volume 4 - pp. 114-128. *Spoken Language and Multimodal Applications for Electronic Realities* 

12/1/2020


Luc JULIA - Patents & Publications


Journal of Knowledge-Based SYSTEMS - 1998 - Volume 10 - Issue 5 - pp. 295-303. *Multimodal User Interfaces in the Open Agent Architecture* 


AI Magazine - Spring 1997 - Volume 18 - Issue 1 - pp. 55-64. *Many Robots Make Short Work* 


Revue Traitement du Signal - Fall 1995 - Volume 12 - Number 6 - pp. 619-626. *Un editeur interactif de tableaux dessines a main levee* 


Conference Papers (44)


044 The PLAY Team. (2001). *MiTV: rethinking interactive TV*. VSMM'2001 : San Francisco (USA). 


043 PRICE P. & JULIA L. (2001). *Using Speech and Language Technology to Coach Reading*. HLT'2001 : San Diego (USA). 


042 HALVERSON C., JULIA L. & BING J. (2000). *How do I get there from here? Exploring speech in CARS*. CHI'2000 : Amsterdam (The Netherlands), Submitted. 

041 HALVERSON C., JULIA L. & CHEYER A. (2000). *Walking on Water: The task of supporting-not coming between-people and work*. COOP'2000 : Sophia Antipolis (France). 


040 TZANETAKIS G. & JULIA L. (2000). *Multimedia Structuring Using Trees*. RIAO'2000 : Paris (France), pp.795-603. 


039 IONESCU A. & JULIA L. (2000). *EMCE: A Multimodal Environment Augmenting Conferencing Experiences*. FAUIC'2000 : Canberra (Australia). 


038 JULIA L. (1999). *Augmenting Humans' Experiences*. OzCHI'99 : Wagga Wagga (Australia), pp 146-148. 


037 JULIA L. & BING J. (1999). *TravelMATE: A demonstration of SRI's Multimodal Augmented Tutoring Environment* IWAR'99 : San Francisco (USA). 


036 JULIA L. & CHEYER A. (1999). *Is Talking to Virtual more Realistic?* EuroSpeech'99 : Budapest (Hungary), vol. 4, pp 1719-1722. 


035 JULIA L. & CHEYER A. (1999). *Facilitating Navigation and Information Access with CANOES: Collaborative Agents and Natural Operations in Enhanced Spaces*. Inter-agency Workshop on "Smart Environment": Atlanta (USA). 

034 JULIA L., BING J. & CHEYER A. (1999). *Virtual Spaces Revive Real World Interaction*. First EC/NSF Advanced Research Workshop "Research Frontiers in Virtual Environments and Human-Centred Computing": Bonas (France), Section 4. 

033 CHEYER A. & JULIA L. (1999). *Designing, Developing & Evaluating Multimodal Applications*. CHI'99 : Pittsburgh (USA), workshop 13: "Designing the User Interface for Pen and Speech Applications". 

032 CHEYER A. & JULIA L. (1999). *InfoWiz: An Animated Voice Interactive Information System*. Agents'99 : Seattle (USA), workshop 6 : "Communicative Agents". 

031 JULIA L. & CHEYER A. (1998). *Cooperative Agents and Recognition Systems (CARS) for Drivers and Passengers*. OzCHI'98 : Adelaide (Australia), pp 32-38. 

030 MYERS G., JULIA L., CHEYER A. & GHERBI S. (1998). *Multimodal User Interface for Mobile Robots*. UIST'98 : San Francisco (USA). 


12/1/2020

Luc JULIA - Patents & Publications

- 029** JULIA L., CHEYER A., DOWDING J., BRATT H., GAWRON J.M., BRATT E. & MOORE R. (1998). *How Natural Inputs Aid Interaction in Graphical Simulations?* VSMM'98 : Gifu (Japan), pp 466-468. 
- 028** DUBREUIL J. & JULIA L. (1998). *Collaborative Use of Multimodal Applications by Lifelike Computer Characters and Humans*. Lifelike Computer Characters '98 : Snowbird (USA) poster session. 
- 027** GUZZONI D., KONOLIGE K., MYERS K., CHEYER A. & JULIA L. (1998). *Robots in a Distributed Agent System*. AAAI'98 (Cognitive Robotics) : Orlando (USA), pp 61-67. 
- 026** KEHLER A., MARTIN J.C., CHEYER A., JULIA L., HOBBS J. & BEAR J. (1998). *On representing Salience and Reference in Multimodal Human-Computer Interaction*. AAAI'98 (Representations for Multi-Modal Human-Computer Interaction) : Madison (USA), pp 33-39. 
- 025** NEUMEYER L., FRANCO H., ABRASH V., JULIA L., RONEN O., BRATT H., BING J. & DIGALAKIS V. (1998). *WebGrader: A multilingual pronunciation practice tool*. STILL'98 : Stockholm (Sweden), pp 61-64. 
- 024** CHEYER A., JULIA L. & MARTIN J.C. (1998). *A Unified Framework for Constructing Multimodal Experiments and Applications*. CMC'98 : Tilburg (The Netherlands), pp 63-69. 
- 023** MARTIN J.C., JULIA L. & CHEYER A. (1998). *A Theoretical Framework for Multimodal User Studies*. CMC'98 : Tilburg (The Netherlands), pp 104-110. 
- 022** CHEYER A. & JULIA L. (1998). *MVIEWS: Multimodal Tools for the Video Analyst*. IUI'98 : San Francisco (USA), pp 55-62. 
- 021** JULIA L. & CHEYER A. (1997). *Speech: A Privileged Modality*. EuroSpeech'97 : Rhodes (Greece), vol. 4, pp 1843-1846.  - Poster 
- 020** JULIA L. & CHEYER A. (1997). *Des Interfaces Multimodales pour Internet*. IHM'97 : Poitiers (France), pp 217-218. 
- 019** JULIA L. & CHEYER A. (1997). *Multimodal Interfaces for Internet*. HCI International'97 : San Francisco (USA), poster sessions, p 117.  - Poster 
- 018** GUZZONI D., CHEYER A., JULIA L. & KONOLIGE K. (1997). *Physical applications of an agent architecture - Examples in mobile robotics and surgery training systems*. SWCD'97 : Lausanne (Switzerland), pp 1-7. 
- 017** JULIA L., CHEYER A., NEUMEYER L., DOWDING J. & CHARAFEDDINE M. (1997). [HTTP://WWW.SPEECH.SRI.COM/DEMOS/ATLS.HTML](http://www.speech.sri.com/demos/atls.html). AAAI'97 : Stanford (USA), pp 72-76. 
- 016** JULIA L., HECK L. & CHEYER A. (1997). *A Speaker Identification Agent*. AVBPA'97 : Crans Montana (Switzerland), pp 261-266. 
- 015** MORAN D., CHEYER A., JULIA L., MARTIN D. & PARK S. (1997). *The Open Agent Architecture and Its Multimodal User Interface*. IUI'97 : Orlando (USA), pp 61-68. 
- 014** JULIA L. (1995). *Interface Homme-Machine Multimodale pour la Conception et l'Edition de Documents Graphiques*. PhD. Thesis : ENST Paris. 
- 013** JULIA L. & CHEYER A. (1995). *A Multimodal Computer-augmented Interface for Distributed Applications*. HCI'95 : Tokyo (Japan), pp 237-240.

12/1/2020


Luc JULIA - Patents & Publications


012 JULIA L. & FAURE C. (1995). *Pattern Recognition and Beautification for a Pen Based Interface*. ICDAR'95 : Montreal (Canada), pp 58-63. 


011


CHEYER A. & JULIA L. (1995). *Multimodal Maps: An Agent-based Approach*. CMC'95 : Eindhoven (The Netherlands), pp 103-113. 


010


JULIA L. & FAURE C. (1994). *Interpretation de tableaux dans une situation d'interaction utilisateur-machine*. CNED'94 : Rouen (France), pp 215-222. 

009 FAURE C. & JULIA L. (1994). *An Agent-Based Architecture for a Multimodal Interface*. AAAI'94 : Stanford (USA), pp 82-86. 

008 POIRIER F., JULIA L., ROSSIGNOL S., & FAURE C. (1993). *Edition de tableaux sur ordinateur a stylo vers une designation naturelle*. IHM'93 : Lyon (France), pp 45-49. 

007 FAURE C., JULIA L., POIRIER F. & ROSSIGNOL S. (1993). *Une interface multimodale pour la conception de documents graphiques*. Rapport d'activite du pole IHMM du GDR-PRC CHM 91-93: Lyon (France), pp 39-49. 


006 JULIA L. & FAURE C. (1993). *A multimodal interface for incremental graphic document design*. HCII'93 : Orlando (USA), poster sessions, p 186. 


005 JULIA L., LEFEBVRE P., POIRIER F. & DUNCAN G. (1993). *User modelling with neural networks in a multimodal Unix environment*. HCII'93 : Orlando (USA), poster sessions, p 237. 

004

FAURE C. & JULIA L. (1993). *TAPAGE : Edition de Tableaux par la PArole et le GEste*. FRI'93 : Palaiseau (France), pp 94-95.

003

FAURE C. & JULIA L. (1993). *Interaction homme-machine par la parole et le geste pour l'edition de documents : TAPAGE*. IMRV'93 : Montpellier (France), pp 171-180. 

002 FAURE C. & JULIA L. (1992). *TAPAGE : Une interface pour l'aide a l'edition de Tableaux par la PArole et le GEste*. IHM'92 : Paris (France), pp 167-171. 

001

JULIA L. (1992). *Vers une interface multimodale*. Rapports de DEA 91-92 : Universite P. & M. Curie, IBP ED 92-1, pp 112-147.

webmaster@lucjulia.com

© 1994-2020 Luc JULIA - All rights reserved